Edomus: Solución libre de domótica personalizada

Sergio Calero Robles
Diego Valbuena Pineda

GRADO DE INGENIERIA INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin grado en Ingeniería Informática

01/09/2019

Director:

Jorge Gomez Sanz

Documento maquetado con LaTeX

Este documento está preparado para ser impreso a doble cara.

Autorización de difusión

Autores

Sergio Calero Robles
Diego Valbuena Pineda

Fecha
01/09/2019

Los abajo firmantes, matriculados en el grado en Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo Fin de Grado: "Edomus: Solución libre de domótica personalizada", realizado durante el curso académico 2018-2019 bajo la dirección de J. Gomez-Sanz en el Departamento de Ingeniería de Software e Inteligencia Artifical, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Prólogo

La domótica es comúnmente asociada al confort en la vivienda: persianas que suben y bajan a golpe de interruptor, luces que se encienden al pasar, equipos de climatización controlados mediante termostatos y una inagotable lista de automatismos, en dispositivos o la propia infraestructura del hogar.

La abundante proliferación de dispositivos electrónicos en el hogar, en las últimas décadas, ha planteado la interconexión de todos estos dispositivos en sistemas centralizados con control remoto. Dicho control, que en el milenio pasado, estaba reservado a una terminal en alguna pared de la vivienda, ahora pueden ser manejados desde un *smartphone* para hacer, por ejemplo, unas tostadas con la tostadora sin tenerla a la vista. El grado de interconexión de dispositivos cada vez es más grande, empezando en hogares y empresas, y terminando en distritos y ciudades. Es por ello por lo que han nacido términos como el $Internet\ of\ Things$ o el $fogging$, para explicar esta red de señales invisibles que nos rodean y que dan vida artificial a los lugares en los que vivimos.

Por supuesto, hay motivaciones más que suficientes para esta aparente necesidad de conectividad de sensores y actuadores. Se ven reflejadas en el nacimiento de la *industria* 4.0 y ya hay quien habla de *humanidad* 2.0. Recolectar ingentes volúmenes de datos para luego realizar estudios conductuales, o previsiones de mercado, son algunas de las aplicaciones más demandadas en nuestra era. La domótica, en ese sentido, puede beneficiarse de estos avances para proporcionar automatismos más allá del confort, como por ejemplo cuidar el medio ambiente y de paso, nuestro bolsillo. Sin embargo, se ha impuesto entre los distribuidores de estas tecnologías, el forzar a los consumidores finales a vender su privacidad y ser fieles a su ecosistema de hardware y protocolos como condición de adquisición.

El interés por conocer el funcionamiento y requisitos reales para una solución de domótica libre es el motor principal de motivación para este proyecto. Se desea averiguar el esfuerzo necesario para crear una solución de estas características y enfrentar las capacidades obtenidas a los largo de la formación como alumnos del grado de ingeniería informática.

Resumen en castellano

Este documento aborda el proceso de creación de una solución domótica integral personalizable, incluyendo una aplicación de móvil para gestionar las habitaciones y los sensores y actuadores ubicados en las mismas, pudiendo recibir información de los dispositivos desplegados, incluir nuevos o reorganizarlos en otras estancias. Se definirán todos los elementos físicos y de software necesarios para la creación de esta solución y se mostrará su implementación, gestión y funcionamiento.

Palabras clave

SmartHome, sistema aislado, Open Software, Open Hardware, privacidad y personalización.

Abstract

This document addresses the process of creating a customizable integral home automation solution, including a smartphone application to manage the rooms and the sensors or actuators located in them, being able to receive information from the deployed devices, include new ones or reorganize them in other rooms. All the physical and software elements necessary for the creation of this solution will be defined and its implementation, management and operation will be shown.

Keywords

SmartHome, isolated system, Open Software, Open Hardware, privacy and personalization.

Índice general

Índice de figuras

Agradecimientos

Es nuestro deseo aprovechar este espacio para agradecer al equipo docente de la Facultad de Informática, pues siempre hemos valorado positivamente el esfuerzo y dedicación que han invertido en todos los estudiantes y con ello se han ganado nuestro respeto y admiración. También estamos en deuda con nuestros más cercanos compañeros de carrera, gracias a los cuales hemos reforzado nuestras aptitudes, superado obstáculos, y encontrado motivación y nuevas energías cuando hicieron falta. Tendréis nuestro respeto y aprecio allá donde estéis. Un agradecimiento especial a nuestros familiares y nuestras parejas, que siempre han estado apoyándonos y velando por nuestros intereses, pese a nuestras diferencias, que ponen de manifiesto la evidente suerte que nos ha tocado en la vida. Y expresar también el orgullo de haber creado este proyecto bajo la tutela de el Dr. Jorge Gomez Sanz.

Aportaciones

Sergio Calero Robles

Responsable del apartado de software del proyecto. Mi aportación al trabajo ha comenzado con unas pruebas de concepto tanto de una aplicación móvil como de un servidor de la aplicación.

Me he encargado de evaluar y proyectar el stack de servicios nos permite crear una aplicación frontend/backend con los requisitos del proyecto. He procurado que la arquitectura del sistema frontend/backend permitiese no sólo el cumplimiento de los requisitos proyectados en este trabajo, sino también mantener el potencial para ampliar el sistema completo tanto verticalmente, con muchas otras funcionalidades y características, como horizontalmente, con soporte para otro tipo de dispositivos.

Me he asegurado de utilizar patrones de diseño adecuados a los requisitos específicos del proyecto, además de procurar cumplir todos los principios de Clean Code y buenas prácticas, buscando siempre una alta eficiencia del código y una alta modularidad y separación de responsabilidades de cada módulo, servicio o componente utilizados. Adicionalmente, en el caso del frontend, he elaborado una estructura de módulos, stores, servicios y componentes que permitiera la reutilización de los componentes web de la forma más clara y eficiente posible.

Con el fin de garantizar un desarrollo de calidad, he establecido metodologías de trabajo adecuadas, como añadir logs de desarrollo claros y útiles en cada paso del código, así como un alto nivel de documentación del código que sea compatible con herramientas modernas de documentación automatizada. Me he responsabilizado de obtener y ofrecer al equipo las herramientas apropiadas para el desarrollo, como entornos de desarrollo, entornos de despliegue y prueba, software para uso de servicios adicionales como BBDD y el conocimiento necesario para un uso óptimo de los mismos.

Me he responsabilizado del diseño y experiencia de usuario de la aplicación móvil, con su consiguiente proceso de testeo de calidad para garantizar un buen nivel de usabilidad y comodidad, con un alto feedback para el usuario de cada flujo que esté teniendo lugar en la aplicación, así como de optimizar una navegación natural e intuitiva.

He llevado a cabo del diseño, construcción y testeo de la API REST que comunica el frontend con el backend, la implementación en ambas partes, así como del diseño de las comunicaciones MQTT entre backend y dispositivos, y su implementación en la parte del servidor.

He contribuido en la documentación de frontend y backend en el presente documento en el capítulo 4, así como la documentación de las fases relativas a ellos en los casos de uso del capítulo 5, además de la elaboración de los diagramas correspondientes y la aportación de algunas conclusiones del capítulo 6.

Diego Valbuena Pineda

Responsable del apartado de hardware del proyecto. Mi aportación al trabajo ha comenzado con una investigación y valoración de adquisición de dispositivos de hardware para la creación del prototipo de este proyecto. Ha sido necesario testear individualmente cada pieza mediante implementaciones individuales para verificar su funcionamiento antes de su puesta en funcionamiento en nuestros despliegues de IoT. Esto incluye la verificación de interoperatividad entre los distintos dispositivos, cuestión la cual ha demostrado ser mas compleja y menos funcional contra mas bajo era el presupuesto definido para minimizar los gastos.

Me he encargo de la selección del sistema operativo que permite la ejecución de aplicaciones. Como extensión, también he procurado la configuración pertinente para establecer comunicaciones remotas seguras con el servidor alojado en el sistema operativo sobre el cual el equipo ha desarrollado los despliegues para pruebas del prototipo. También incluye el mantenimiento del software, creación de cuentas, gestión de permisos en directorios y descarga de versiones de paquetes adecuados para el software necesario para las pruebas de hardware con los nodos y el gateway.

El desarrollo de los apartados relacionados con el hardware y el estudio del estado del arte de los mismos. Así como la explicación e implementación del script generador de código para placas microcontroladoras definidos en el capitulo 4 que permite tostar el firmware personalizado segun argumentos en los nodos. También se me ha permitido condensar las ideas mías y de mi compañero expuestas en el capitulo 1 y el apartado de conclusiones y mejoras a futuro.

La valoración, experimentación y decisión de uso de protocolos de comunicación inalámbrica. Determinar el grado de dificultad de desarrollo que implicaba cada opción de las estudiadas, así como su rendimiento, consumo de batería y facilidad de uso por parte de la aplicación. Esto incluye su implementación de código dentro de las placas microcontroladoras y el despliegue de red del gateway como punto de acceso para los nodos.

La reparación o sustitución de componentes dañados, concretamente actuadores como leds, cableado, o sensores que por cuestiones de movilidad, montaje incorrecto o manipulación descuidada han terminado sufriendo desgate o ruptura.

El mantenimiento de servicios de terceros para la redirección DNS y configuración de enrutamiento para que el servidor estuviese disponible en la red de internet.

La experimentación de procesos de compilación cruzada y subida de código en los nodos mediante compilación con ficheros make y entornos de desarrollo CLI. Coordinación con las pruebas de software para unificar la vertiente de hardware y software en los puntos necesarios, siguiendo las indicaciones de mi compañero para facilitar la estructura de mensajes que mejor se acomodase a las necesidades de la aplicación.

Los diseños de impresión 3D y su posterior creación para reforzar la integridad del nodo principal en sus múltiples desplazamientos para las pruebas de funcionamiento en diferentes entornos.

"Qué hermoso es hablarle a la máquina en su propio idioma, y que nos responda en el nuestro"

Cid Meier's Civilization®: Beyond Earth

Capítulo 1

Introducción

1.1. Motivación

La domótica ha sufrido un crecimiento acelerado en los últimos años gracias a la interconexión de dispositivos electrónicos gestionados con aplicaciones móviles. Este fenómeno social de tener una APP para controlar la casa se basa en la venta de kits de domótica Plug and Play. Estos requieren únicamente de la instalación de una aplicación de móvil gratuita y la adquisición de los productos ofertados por los fabricantes que operen en los ecosistemas de dichas aplicaciones. Una gran competencia entre empresas ha surgido a raíz de este planteamiento, ofreciendo una gama extensa de electrodomésticos que pueden combinarse, en algunos casos, incluso interoperar entre distintas marcas. Esta disputa se está desarrollando en una etapa de incertidumbre, causada por la fase experimental de productos que se ofertan a los consumidores, ya que aún no existe una necesidad real de domótica en las personas, como ocurre, por ejemplo, con un smartPhone.

Se siguen buscando estrategias de marketing para crear dicha necesidad mediante productos que aportan confort o gestión remota de electrodomésticos del hogar. Tomemos por ejemplo los sistemas de iluminación con múltiples configuraciones de intensidad, o el tracking de actividad, los históricos de peso en una báscula, etc. Independientemente de las ideas presentadas, ya se están estableciendo unas pautas comunes en todos los actores del

sector de la domótica. Y es en estas pautas donde aparece nuestra preocupación a la hora de optar por las soluciones con más presencia del mercado (véase Amazon, Google o Xiaomi), que motivan en este proyecto la búsqueda de una suite de domótica que se aleje de sus prácticas y tendencias. A continuación, se describen los puntos principales que serán objeto de análisis, y donde se encuentran problemas que han sido creados, para beneficio de las marcas, y en detrimento del usuario final que los adquiere.

En primer lugar, la estrategia es ofrecer dispositivos de distintos rangos de precio, que están diseñados para utilizarse de forma exclusiva en los ecosistemas de cada marca. Esto implica supeditarse a las imposiciones técnicas de cada fabricante, incluyendo la forma en la que funcionan dichos dispositivos, sin poder modificar las especificaciones y funcionamientos de los mismos. En esencia, cajas negras, que desconocemos su funcionamiento interno. Esto se extiende a la posibilidad de ampliar sus funciones o repararlos por cuenta propia (tal y como se recoge en las cláusulas de uso definidas en los manuales de todas las marcas). Se deja al usuario final a merced de un contrato establecido con los fabricantes, incluyendo su soporte de post-venta y servicio técnico. Por escenificarlo en un ejemplo análogo, si se adquiere un coche, no se impide que se pueda arreglar dicho coche con piezas genéricas, ni que se esté obligado a repararlo en la casa oficial de la marca. Esta tendencia, por suerte, parece estar empezando a diluirse, y es más fácil encontrar ahora dispositivos que funcionan entre múltiples plataformas simultáneamente y son mas abiertos.

En segundo lugar, preocupa la aceptación de las políticas de uso y privacidad de las aplicaciones de las distintas plataformas, que obliga a un uso restringido de los productos según las pautas del vendedor, así como la cesión de la privacidad del usuario final. Toda interacción del usuario con los dispositivos, así como otros datos personales obtenidos en procesos de registro de sus aplicaciones, incluyendo Datos de carácter personal (DCP) de carácter bajo, medio, y alto según qué dispositivos, serán monitorizados y enviados a servi-

dores para ser procesados, sin tener claro con qué fin, salvo de que operen en el territorio europeo donde la Reglamento general de protección de datos (RGPD) trata de regular estas situaciones.

Toda esta transferencia de datos, en todas las plataformas, está planteada para usar una solución domótica en servicios en la nube, lo que implica un envío continuo de datos a servicios externos, con medidas de seguridad desconocidas, sin garantías reales de protección de datos, y bajo la eterna dependencia del funcionamiento de dichos servicios, que operan en países con leyes que podrían no estar en sintonía con la legalidad del país en el que vive el usuario final. Los riesgos de seguridad en Internet de las Cosas (IoT) tienen inpactos importantes cuando se materializan, los dispositivos que conforman un despliegue pueden verse involucrados en ataques informáticos con el fin de incorporar dispositivos a una `bootnet`, uno caso reciente que obtuvo gran repercusión mediatica y mayor impacto en la industrias del IoT fuela red Mirai, tal y como se documenta en el articulo 'DDoS in the IoT: Mirai and Other Botnets'[KKSV17]. También hay que tener en cuenta que todas estas empresas se reservan el derecho unilateral de cambiar sus condiciones de uso y política de privacidad cuando crean convenientes sin consentimiento del usuario final. Y merece una mención especial que, las pruebas ejecutadas en las aplicaciones móviles de los casos que se analizarán en el capítulo2 coinciden en una misma máxima: Cuanto más restrictiva sea la configuración de privacidad por parte del usuario, menor será la utilidad de la aplicación.

Y en último lugar, como motivación adicional para el desarrollo de una solución domótica independiente y libre, se encuentra el problema que bautizaremos como "Miles de kilómetros por metro", es decir, que para encender una interruptor a un metro del usuario, la operativa de la infraestructura necesaria para que la acción ocurra implica que los datos de dicha acción tienen que viajar miles de kilómetros desde el móvil del usuario, hasta llegar a los servidores necesarios, para algo tan trivial como encender un interruptor. Esto carece de

justificación, salvo de que exista la intención de usar los datos del usuario para nutrir dataset para Machine Learning o para su procesamiento en estrategias de marketing o dotar de tracking a empresas de terceros, además, en caso de caída de la red de internet (ya sea por el Proveedor de servicios de internet (ISP), por un bloqueo del servicio en origen o destino, etc.) no se puede operar con normalidad los dispositivos en la red local del hogar, lo cual no está justificado. Es entendible que, ante desconexión de la red de internet no se pueda operar de manera remota la domótica del hogar, pero un router doméstico puede seguir ofreciendo operatividad a la red local, incluyendo la gestión domótica desde dentro de la red, para cubrir la funcionalidad básica.

Por todo esto, el objetivo es investigar y crear un prototipo de solución domótica (*suite domótica* en adelante) que evite estas pautas anteriormente descritas. Sustentado por un software libre, con unas librerías accesibles de manera pública, basado en un hardware fácil de adquirir y que con una base fundamental de conocimientos de electrónica e informática, dicha suite pueda ser manipulada y personalizada con relativa facilidad. Una solución integral de domótica que incluya el software necesario para operar localmente en casa y externamente a través de la red de internet, con una APP, que no precise de servicios de terceros, ni de la aceptación de políticas de uso privativas, ya que el responsable final de los datos sera el propio usuario.

En el capítulo 2 se analizarán las distintas soluciones existente para las distintas capas físicas y de software, y se valorarán cuáles se acercan más a los planteamientos en la sección de objetivos 2.3. Por otro lado, para entender qué tan complejo es crear una suite de domótica libre desde la base, y comprender las técnicas e implementación de cada etapa, esta documentación estará apoyada en la creación un prototipo funcional.

1.2. Objetivos

Diseñar e implementar una solución integral de domótica modular, con un proceso sencillo de agregación de dispositivos a dicha suite que sea flexible ante escenarios no previstos.

- Investigación, selección, instalación y configuración de un conjunto de servicios que permitan controlar la suite de domótica desde un servidor físico que actuará como nodo principal.
- Desarrollo de una aplicación móvil que permitir al usuario interactuar mediante una API-REST con dicho servidor para ejecutar las acciones y configuraciones, que solicitando el número de permisos más básicos demuestre que el exceso de permisos solicitados por las marcas responde sólo a interés empresariales y no funcionales.
- Desarrollar un sistema de adhesión de dispositivos a la suite domótica mediante un wizard en la aplicación móvil.
- Aplicar la solución en un escenario simulado para monitorizar el ambiente de una bodega y controlar el sistema de luces en las escaleras de acceso de forma remota, todo ello mediante una aplicación de móvil.
- Mantener el desarrollo de un prototipo dentro de un rango de precio considerado bajo.

1.2.1. Alcance

No se pretende crear una solución disruptiva en el mercado de la domótica, que enfrente las soluciones ya existentes ofertadas por otras marcas, ni aportar un protocolo nuevo de comunicaciones entre dispositivos IoT. Principalmente es conocer los requisitos necesarios para la creación de una suite domótica y la capacidad necesaria de un usuario final para que pueda instalar su propio sistema de domótica personalizado, sin depender de servicios a terceros.

Las soluciones que pueden adquirirse actualmente en el mercado, tras lo años de prueba y error por parte de las marcas, han alcanzado un proceso de instalación muy sencillo para los usuarios finales. Esto, sin embargo, será difícil de abordar en este proyecto, ya que será necesario que el usuario disponga de conocimientos específicos de informática para interpretar los pasos que estarán documentados, pero es posible alcanzar un punto intermedio, que requiera de unos conocimientos moderados, pero que este lo suficientemente guiado y configurado como para ser trivial. Se intentará minimizar, en la medida de lo posible, todos los pasos necesarios para montar la infraestructura, incluyendo la instalación de software y la programación de script, para que el prototipo pueda ser exportado con facilidad y replicarse nuevamente ahorrando tiempo, y simplificando el proceso.

Crear una suite domótica genérica puede cubrir la vertiente de aprendizaje, pero se quiere aportar algún ejemplo de cómo esta generalidad puede cubrir necesidades de casos concretos. Esta suite debe proveer de toda la arquitectura necesaria para que su flexibilidad de configuración pueda cubrir casos de uso concretos no previstos. Se ha decidido probar el prototipo de la suite de domótica resultante de este proyecto para ayudar con el control de ambiente de una bodega situada en el sótano de una casa, y el sistema de iluminación de dicha bodega.

1.3. Plan de trabajo

El diseño de una suite de domótica, aun creándose desde cero, debe aprovechar al máximo las tecnologías y desarrollos de software libres existentes, ya que éste es precisamente el mayor potencial del desarrollo colaborativo tan característico del software libre, que incluyen una extensa comunidad que día a día mejoran el rendimiento y seguridad de cada uno de los módulos que puedan componer este proyecto. Esto implica un estudio previo de las diferentes opciones disponibles, y una selección del software y hardware que mejor se ajuste a nuestros objetivos, más detallados en el capitulo 3. Podemos separar las distintas fases de la siguiente forma:

1. Investigación de software: Incluye una evaluación de la disponibilidad de software que cubra las especificaciones que deseamos tener. Será necesario verificar si para cada idea de implementación ya existe una solución, y en caso de existir, valorar si merece la pena crear una implementación propia (por cuestiones de aprendizaje, versatilidad o adecuación).

2. Investigación de hardware: Determinar el marco de opciones clásicas utilizadas al prototipar sistemas de IoT. Entender los rangos de precio de los mismos y su escalabilidad y protocolos de funcionamiento. Al no crearse hardware nuevo, se seleccionarán las opciones que mejor encajen con los objetivos del proyecto.

3. Experimentar y prototipar: Aquellas tecnologías que sean reutilizadas deben poder conectarse entre sí, con armonía, y facilidad, siendo, en aquellos puntos que sea necesario, ajustar las configuraciones e incluir desarrollos propios que permitan a todas estas tecnologías operar como un único sistema. Implementar las soluciones propuestas para cada vertiente del proyecto, en un prototipo global, que cubra todos los objetivos propuestos. Iterar el diseño de cada módulo de manera paralela e independiente, evitando que las dificultades aisladas no bloqueen el desarrollo del resto de módulos.

1.4. Estructura del documento

El documento se estructura como sigue:

- El capítulo 2 evalúa la actual situación tecnológica y planteamientos de desarrollo disponibles para crear una suite domótica libre.
- El capítulo 3 se centra en la definición de propuesta para crear un prototipo de la suite de domótica, así como funciones y diagramas de despliegue.
- El capítulo 4 contiene el diseño de una arquitectura IoT con aplicación móvil para operar la suite, así como el criterio de selección de equipamiento, despliegue de hardware y el diseño del software para la aplicación móvil.
- El capítulo 5 propone casos de uso, donde el prototipo opera y da solución a los problemas planteados.
- El trabajo concluye en el capítulo 6 con unas reflexiones sobre el trabajo hecho y unas líneas de trabajo futuro.

Capítulo 2

Estado del Arte

Como definición, el término 'domótica' proviene de la unión de las palabras `domus` ('casa' en latín) y `-tica` (de 'automática', palabra en griego que significa 'que funciona por sí sola'). De manera más técnica y extensa se puede expresar por domótica, al conjunto de sistemas que hacen de una vivienda un edificio inteligente, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas, y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar. De esta definición no se desglosa nada que concuerde con los problemas descritos en el capítulo 1.1.

'Para que una casa funcione por sí sola', no parece estrictamente necesario que, lo que la casa hace, lo tenga que saber y gestionar una persona ajena a la casa, más concretamente, una organización. Si los datos generados, que definen cómo la gente usa el sistema son enviados y procesados fuera del hogar, no existen garantías plenas de que una persona ajena a la unidad familiar pueda visualizar lo que ocurre dentro, desde fuera. No se busca demonizar a las compañías que ofrecen soluciones domóticas, pero sus practicas exponen a las personas a estos riesgos. En cuanto a funcionar de forma autónoma, se podría entender que un sistema es autónomo si, independientemente de factores externos al sistema, éste puede seguir realizando sus funciones fundamentales de manera normal e indefinida en el tiempo. Pero si la solución pasa por un servicio externo y no por las capacidades de la propia

casa, eso significa que este sistema no es capaz de operar por su cuenta propia si el proveedor de servicio externo falla.

Sobre esta dependencia de servicios en la nube surge el paradigma de `edge comupting`, donde los datos registrado por sensores y acciones de actuadores de dispositivos de IoT, son procesados en niveles más cercanos al lugar donde ocurren en lugar de ser enviados por la red de internet a centros de procesamiento. La estrategia de delegar el procesamiento en los niveles más bajos de un despliegue de IoT, en lugar de subir toda la información a la red para ser procesada y luego recibir la respuesta, surge como solución a varios problemas fundamentales del procesamiento en la nube. En la figura 2.1, extraída del articulo 'Internet of Things (IoT): A vision, architectural elements, and future directions'[GBMP13], puede observarse un despliegue de IoT cuyo procesamiento de datos esta centralizado en la nube.

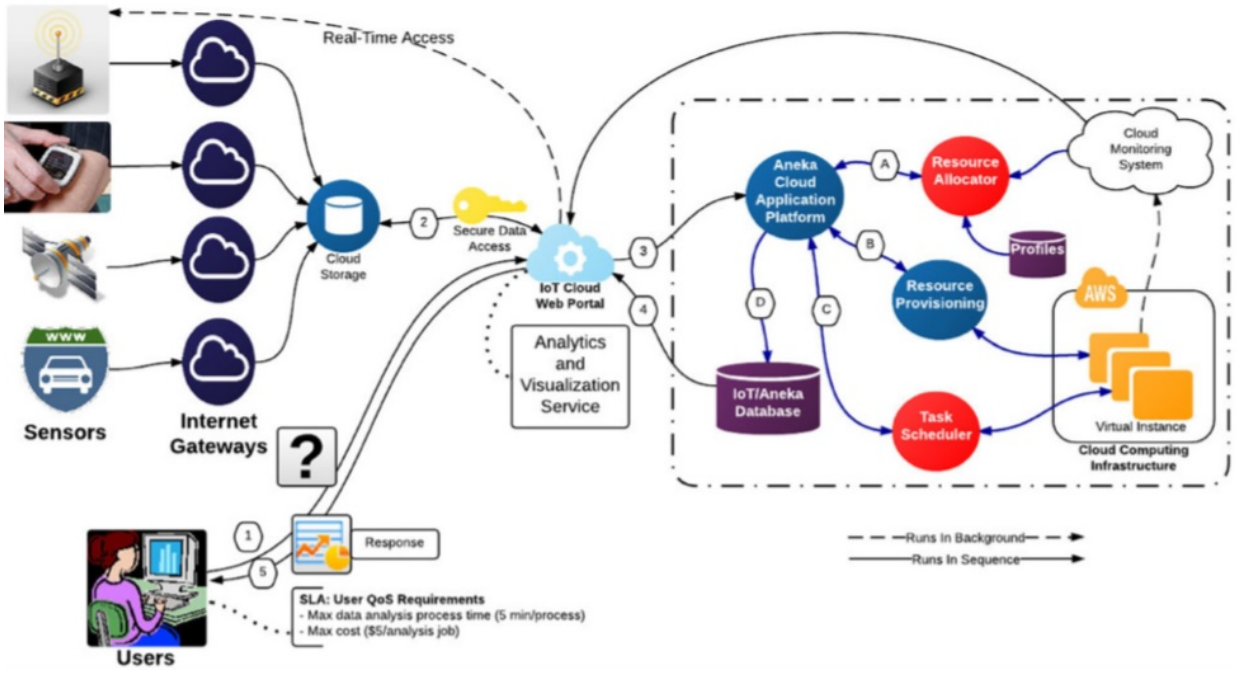

Figura 2.1: *Un modelo de interacción end-to-end entre varios actores en un framework de nube centralizado*

Un primer problema de este esquema es la latencia. Si un elemento sensor que actúa como disparador de un elemento actuador debe esperar a que la información del sensor sea enviada a la nube, procesada para evaluar si dicha información implica que se active el actuador, y enviar de nuevo desde la nube dicha señal al actuador, en todo este proceso se genera un espacio de tiempo, que según que circunstancias puede no ser viable, por ejemplo, en un coche autónomo el sistema de frenado cuando va a colisionar con un peatón. A nivel doméstico, otro ejemplo que requiere de un tiempo de respuesta mínimo podría ser una puerta que se cierra mientras que en el proceso una persona pone su mano en el marco y un sensor de presión detecta que debe interrumpir el cerrado. Estos retrasos de respuesta producidos por la latencia son asumibles en la mayoría de los casos de uso de dispositivos de domótica ya que no requieren esta inmediatez de proceso.

El segundo problema de los esquemas IoT centrados en la nube, es una cuestión energética. Los centros de procesos de datos, donde se almacenan las plataformas y servicios que gestionan los despliegues de IoT requieren de infraestructuras de suministro eléctrico ya que ejecutan servicios basados en internet de en una enorme escala. Si el proceso de datos se puede hacer en las fuentes del origen de los datos, conocidos como nodos, o en los concentradores de dichos `nodos`, conocidos como `gateways`, puede reducirse la cantidad de potencia de procesamiento requerida en los servidores finales. Delegar el procesamientos de datos, su almacenamiento temporal o la toma de decisiones en los flujos de ejecución de un proceso en los niveles inferiores de un despliegue de IoT distribuye el consumo energético a lo largo del despliegue en vez de concentrar todo el proceso (así como la mayor demanda energética) en un único punto. El distribuir la energía no reduce la cantidad total de energía que se usa en un despliegue de IoT, sin embargo, los elementos inferiores del despliegue como los nodos, pueden en su mayoría estar programados en muchas circunstancias para entrar en modos de hibernación, o directamente apagarse cuando no son necesarios, reduciendo el consumo, y eso incluye a la cantidad de proceso que la nube ha delegado en ellos. La estrategia de

`edge computing` ayuda a esta distribución de energía como se explica en el articulo 'The Promise of Edge Computing' sobre las ventajas del procesado de datos en `network edge` en lugar de completamente en la nube [SD16].

Sobre el planteamiento de despliegues de IoT centralizados en la nube, también se debe considerar los costes de ancho de banda de la información trasmitida a través de la red. Cada protocolo de comunicación dispone de ventajas que les hacen más aptos para cada escenario. En la capa de trasporte definidas en el RFC 1122 de 'Requirements for Internet hosts' [Bra89] encontramos dos protocolos que abordan la transferencia de información mediante cabeceras de confirmación como es el caso de Transmission Control Protocol (TCP) o con un envío sin confirmación como User Datagram Protocol (UDP). Mientras que el primero es muy verboso, y manteniendo una confiabilidad en los mensajes enviados, pudiendo asegurarse la integridad de los contenidos de los datagramas enviados en los paquetes, y reenviando dichos paquetes en caso de perdida durante el envío, esto genera un mayor requisito del ancho de banda necesario. UDP en cambio es más ligero al desprenderse de estas cabeceras de verificación; por supuesto, esto genera habitualmente que se pierdan paquetes en los envíos, pero no representa un problema si el flujo de información es constante en el tiempo y la pérdida puntual de información no tiene un impacto real en la comunicación. Suponiendo un sensor que toma medidas de manera continua sobre un parámetro concreto, se puede obviar puntualmente fallos en la comunicación si seguidamente tras el fallo se suceden datos correctos. Esta estrategia es aplicada habitualmente en tecnologías de streaming, donde la pérdida de un fotograma es apenas apreciable por el usuario que visualiza el contenido de un vídeo.

En escenarios donde una comunicación de un despliegue IoT requiera de seguridad adicional, los protocolos de transferencia como HyperText Transfer Protocol (HTTP) o Message Queuing Telemetry Transport (MQTT) requieren del uso del protocolo de transporte TCP,

ya que las cabeceras adicionales que incluye permiten que la comunicación se establezca con garantías de confiabilidad y técnicas de seguridad como cifrado en los mensajes enviado, verificación de envío y recepción y no repudio. Esto tiene un efecto en la congestión del ancho de banda de las redes de comunicación y su aplicación debe considerar el impacto en los canales de comunicación que supone el uso de un protocolo como TCP que es más pesado que el protocolo UDP. En el documento 'Throughput Analysis and Measurements in IEEE 802.11 WLANs with TCP and UDP Traffic Flows' [BCG07], los resultados de pruebas indican que el rendimiento total del protocolo TCP es independiente del número de conexiones abiertas, y su tráfico agregado se modela como dos flujos saturados, mientras que el del protocolo TCP, para una cantidad n de flujos, obtienen casi n veces el rendimiento agregado logrado en los flujos TCP. Considerando esta conclusión, el protocolo UDP es más apto para un despliegue multitudinario de dispositivos que el protocolo TCP. Si bien es cierto que, en domótica, es difícil que exista un número extenso de dispositivos conectados simultáneamente en la red del hogar.

Y uno de los problemas cuyo impacto posee tanto una vertiente legal que tecnológica es la privacidad de los datos enviados a la nube. En este punto en particular, los despliegues de IoT basados en procesamiento en la nube, son los que poseen más riesgos, ya que al viajar los datos a través de la red de internet, están más expuestos a la interceptación por terceros. En domótica, los datos transferidos por la red están relacionados con los objetos cotidianos equipados con sensores y actuadores que permiten adquirir información sobre los usuarios de manera automática, permitiendo la recopilación de datos que conduce a inferencias sobre la persona monitorizada. Esto permite crear perfiles de consumidor mediante la denominada técnica de "fusión de sensores". Los datos obtenidos en despliegues de domótica pueden parecer datos no personales, pero permiten definir el comportamiento de una persona y crear identidades virtuales que son usadas en estrategias de marketing por las empresas de consumo [Mur]. En el capítulo 1.1, se mencionan tres marcas que ofrecen suites de domótica

a nivel internacional. Se trata de actores con gran presencia en el sector de la domótica de consumo para el hogar. Su importancia es tal, que están definiendo con sus productos el futuro de la domótica, y todas ellas, en mayor o menor media, se ven envueltas en escándalos sobre el tratamiento de datos de usuarios. En el siguiente apartado, profundizaremos en cómo estas empresas abarcan la normativa impuesta en Europa y los estados miembros que recientemente ha obligado a muchas empresas a adoptar su marco de actuación para adecuarse a la norma del RGPD.

2.1. Políticas de privacidad de marcas con gran presencia de mercado

Todo lo que gira en torno al concepto 'política de privacidad' o 'términos de condiciones de uso', se puede traducir, como la experiencia nos ha enseñado, a una aceptación a ciegas de los contratos por parte del usuario final. Esta causa viene motivada principalmente por la complejidad de obtener, abstraer y entender los extensos puntos que conforman estos textos legales, que siguen estando muy lejos de ser amigables. Observamos como se presentan dichos textos en las siguientes marcas, para entender las dificultades que un usuario debe enfrentar si quiere saber exactamente qué está contratando.

`Amazon Alexa (Amazon Movile LLC)`: Su política de privacidad [Ama19c] indica que se suben grabaciones del usuario a sus servidores para ser procesadas en sus sistemas de reconocimiento de voz y compresión de lenguaje. No se indica expresamente que dichas grabaciones, además, son incluidas en procesos de entrenamiento de Machine Learning para mejorar su aplicación. Se indica además, que su dispositivo no graba continuamente, salvo por reconocimiento del comando de activación (generalmente el nombre del dispositivo), pero tampoco se especifica cuánto tiempo graba, ni qué hace con las filtraciones de voz de otras personas que estén presentes. La cláusula de servicios a terceros resulta bastante ambigua, pero indica que cualquier cesión de permisos a una aplicación de terceros que se integre

con Amazon, podrá requerir datos a la misma. Se pueden encontrar más detalles en sus condiciones de uso [Ama19a]. En la misma podemos observar adicionales cláusulas anidadas que redirigen a nuevos enlaces como las condiciones de uso de Amazon España [Ama19b]. Al final es realmente difícil saber a qué se atiene un usuario que adquiere este producto. Su aplicación de móvil también requiere un abanico realmente extenso de permisos en el Sistema Operativo (SO) de un Smartphone/tablet, incluyendo gestión de cuentas en el dispositivo (añadir o eliminar cuentas), accesos a contactos, ubicación, mensajería, llamadas, almacenamientos, dispositivos de entrada/salida y un largo etcétera que puede consultarse en la tienda de aplicaciones correspondiente de Android/iOS.

`GoogleHome (Google LLc.)`: Google lleva años intentando mejorar sus servicios respecto a las leyes europeas de protección de datos. Aun habiendo sonados casos de tratamiento ilegal de DCP que han terminado en sanciones a la compañía, como las de la Agencia española de protección de datos (AEPD) que alcanzan hasta 900.000 euros tal y como recogía en esta noticia del medio [j.m14], aun así, eso no ha frenado la expansión de la compañía a lo largo del mundo. Todo ello seguramente condicionado a su presencia internacional, ya que en algunos países son además proveedores de servicios e infraestructuras de instituciones públicas como ocurre en la `Universidad Complutense de Madrid`, así como proyectos conjuntos como la digitalización de la Biblioteca Complutense [UCM14]. En el campo de la domótica, lo que respecta a su política de privacidad en su aplicación `Google Home`, se remite al usuario a las condiciones generales [Goo19b] de la compañía de `Google LLc`, para información más concreta sobre el producto, es necesario buscar información en la página de ayuda de Google [Goo19a]. En la misma, se afirma la intención de recoger datos (no se especifican de qué naturaleza), con el fin de usarse en Machine Learning. Se confirma que podrían existir aplicaciones de terceros que nutran a `Google` con más datos (tampoco se especifican qué datos ni qué aplicaciones de terceros). Aun así, ofertan la posibilidad de gestionar los datos almacenados en su gestor de actividad y en caso de tener la configuración

por defecto de recogida de datos que `Google` utiliza en los SO de Android, los resultados son, como poco, inquietantes. Un sistema de domótica de esta compañía es perfectamente capaz de determinar con exactitud casi toda la vida diaria de una casa, incluyendo cuándo se duerme, cuándo se come, quien esta en el hogar, y qué actividades se han desarrollado.

`Xiaomi Mi (Xiaomi INc.)`: La empresa china que ha irrumpido en el mercado europeo gracias a sus aplicaciones con interfaz amigable, presentaciones de productos semejantes a las ejecutadas por compañías como `Apple` y precios competitivos, que además admiten múltiples dispositivos clónicos de fabricantes no adscritos a la marca de `Xiaomi` también está haciendo sus adaptaciones para operar en Europa bajo el marco de la RGPD: su política de privacidad [Esp19] sobre la recopilación de datos es, cuanto menos, ambigua, usando frases como, "podemos recopilar la totalidad o una de la parte que usted nos proporciona", y en lo que respecta a de qué manera se utiliza dicha información, es tan extensa que sólo remarcaremos que aparece el nombre de la compañía `Facebook` en uno de los puntos. Se afirman en que no se venderán los DCP a terceros salvo aceptación del usuario. La estrategia de la compañía de `Xiaomi` en su aplicación pasa por ofertar toda clase de servicios, incluyendo chats, foros, pagos en la plataforma de AliPay, etc. La consecuencia de estos servicios genera la tabla más larga de permisos necesarios en un smarthpone/tablet de los ejemplos mencionados. Definitivamente, aquel usuario que instale esta aplicación y permita todas esta condiciones puede olvidarse de su privacidad.

El uso extensivo en todas las marcas de un término tan general como Machine Learning es ambiguo, ya que en ninguno de los casos se especifica qué tipo de modelos analíticos se pretenden crear, aunque se menciona que el objetivo es mejorar sus servicios. Bajo este argumento, un usuario puede entender que ha pagado por un producto una cantidad concreta de dinero, pero el uso de ese producto, generara unos beneficios para la empresa que se verán de nuevo retribuidos en el usuario con un mejor servicio. Pero existen mejoras que

serán exclusivas para la empresa y no repercutirán en el usuario de forma directa, puede que de ninguna forma incluso, como ocurre con modelos analíticos de datos orientados a marketing, que facilitarán a la empresa la capacidad de expandir su negocio y beneficiarse. No es que este intercambio de beneficios sea incorrecto, pero se inclina a favorecer a la empresa sobre el usuario en la mayoría de los casos. En general toda plataforma domótica tratara de nutrir dataset con técnicas de Machine Learning para mejorar la calidad de servicio; aun así, existen plataformas basadas en licencias de código abierto o licencia libre que permiten al desarrollador determinar si dicho flujo de datos puede usarse para este fin o deben permanecer en manos de los usuarios finales. En el siguiente apartado 2.2 se enumeran algunos framework comunes en proyectos de IoT y cómo concuerdan con esta preocupación sobre los DCP.

2.2. Frameworks disponibles para la gestión de IoT para SmartHomes

Un planteamiento recurrente en el diseño de una solución basada en software es acelerar el proceso de desarrollo e implementación utilizando un framework. Es una buena idea. Estas herramientas están, en su mayoría, profundamente documentadas para exprimir sus capacidades al máximo, disponen de versionados y revisiones (en mayor o menor medida) que fortalecen tanto su seguridad, robustez e implementación. Poseen una buena abstracción del hardware en el que se ejecutan, sus servicios son modulares y su arquitectura es escalable. En ocasiones están basados en software libre y/o gratuito, y disponen de una comunidad activa de usuarios a los que poder exponer dudas. Éstas son cualidades muy importantes, más allá de las capacidades técnicas que cada opción pueda ofrecer.

En despliegues de IoT es importante determinar el alcance en conectividad que se desea alcanzar y volumen de datos a tratar. Existen framework pensados para interconectar ingentes cantidades de dispositivos en grandes extensiones de terreno y bajo el peso de un

abrumador volumen de datos que procesar como es el caso de las SmartCities, o la infraestructura del sector primario y secundario. En algunos casos, el alcance es tan extremo que el concepto de IoT evoluciona a Internet of Everything (IoE) y se requiere la presencia de grandes actores tecnológicos, y sus soluciones, para abarcar estos proyectos, como es el caso del framework `IBM BlueMix` de IBM, el `Cisco Virtualized Packet Core` de Cisco, `AWS IoT` de Amazon o `Azure IoT` de Miscrosoft, por mencionar algunos ejemplos de este calibre. Estos framework no fueron diseñados pensando en reducidos entornos como los de un hogar, y aunque son compatibles, el tiempo necesario en formación para su uso queda fuera de las capacidades y expectativas de un proyecto de las características que aquí se recoge. Sin embargo, también se dispone de un amplio abanico de opciones a un alcance más acorde a lo esperado de una solución domótica.

Antes de realizar cualquier evaluación sobre las bondades de cada plataforma, se debe recordar que en la solución a buscar se intenta evitar el uso de servicios en la nube o dependencias de Application Programming Interface (API)s externas. Esto responde al objetivo de aislar la suite domótica a desarrollar, de la red de internet, evitando esa dependencia para su operatividad. Por supuesto, no es el objetivo crear una plataforma desconectada, ya que se espera poder operar de forma remota los dispositivos desde fuera del ámbito de la red local del hogar. Además, debe disponer de un licenciamiento de código libre y gratuito. Y no menos importante, están los costes económicos que deben asumirse al crear el prototipo.

`FIWARE`: Está catalogada como una plataforma de código abierto que agrupan un set de estándares universales para el contexto de gestión de datos [FOUa]. Se sustenta en la ejecución de un framework ejecutado sobre un docker, que puede ser alojado localmente en un ordenador dentro del hogar, y aunque esta solución está orientada a procesar datos en un contexto más extenso que una casa, puede aislarse de la red de internet. Limitan la portabilidad de las aplicaciones a aquellas que se catalogadas 'Powered by FIWARE', y

aunque ofrecen una interfaz estándar para los componentes que integren la solución, con el objetivo de eliminar el bloqueo del proveedor de componentes, posee un licenciamiento de código abierto y un entorno de ejecución no comercial llamado `FIWARE Lab` que permite experimentar sus tecnologías. Estos laboratorios de pruebas están instalados en dockers propiedad de la propia compañía de `Fiware` y es ejecutado sobre una instancia de `Open Stack`. Para desarrollar cualquier tipo de despliegue IoT basado en esta tecnología, es obligatorio integrar en el desarrollo de la aplicación el llamado 'Orion Context Broker Generic Enabler', el cual habilita la gestión de información contextualizada de manera descentralizada y a gran escala.

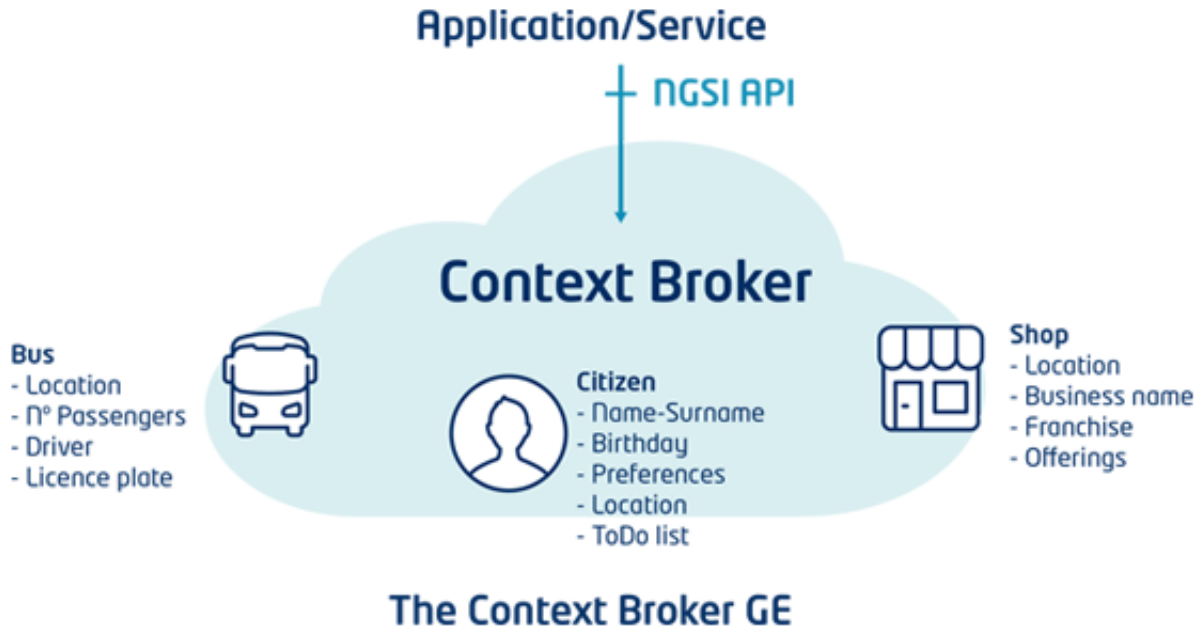

Figura 2.2: *Broker de contexto de información de Fiware*

Facilita el API restful `FIWARE NGSIv2` que permite ejecutar queries, actualizaciones o suscripciones a cambios de contexto de la información. Este contexto de la información permite realizar consultas sobre sensores en distintos escenarios en función de un mismo criterio, tal y como se muestra en la figura, en la cual la misma consulta puede ser ejecutada

en varios escenarios definiendo el contexto adecuado. Esta estrategia no permite explotar su potencial en un proyecto centrado en la domótica de un hogar, que no está planteado para beneficiarse de contextos de datos recogidos por sensores ajenos al propio hogar.

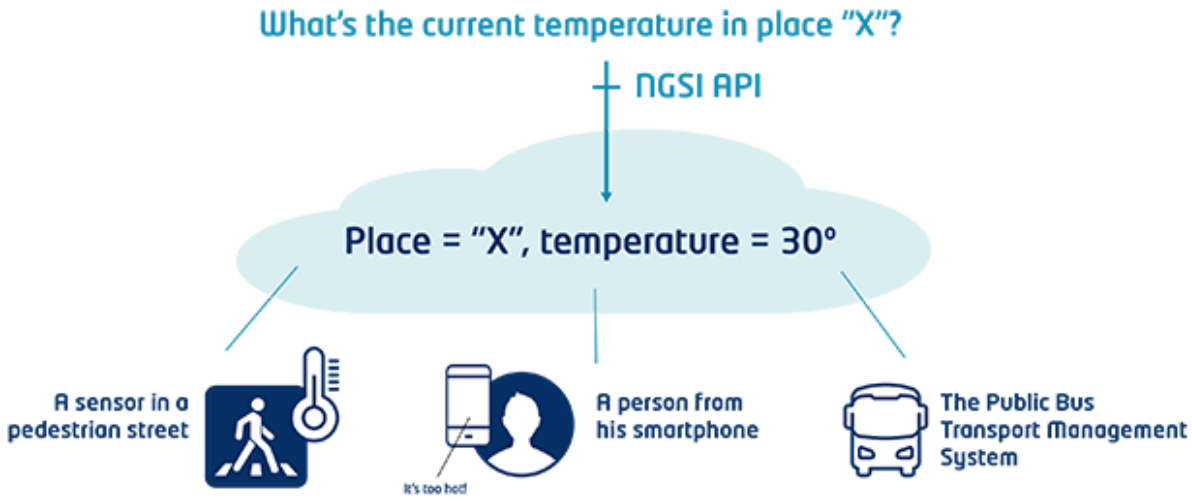

Figura 2.3: *Acceso a la información en múltiple contextos con Fiware*

`OpenHab`: Recomendado frecuentemente en foros y ponencias de IoT, bajo el nombre de `Open Home Automation Bus`. Esta plataforma dispone de manuales de instalación para convertir un ordenador en un centro de control de domótica, incluyendo la propia Raspberry Pi con una imagen ya configurada [oC]. La documentación detalla la definición de un modelo de desarrollo orientado a objetos, flexible y escalable, donde las llamadas `things` representan a los dispositivos físicos, que incluyen las propiedades para gestionar sus canales de comunicación así como `items` que representan la capacidades y propiedades de los automatismos del hogar. También se proponen reglas, que definen comportamientos en función de los disparadores asignados por el usuario. De esta forma, puede automatizarse el apagado de las luces de una estancia si en ésta no hay individuos. Dispone de un extenso abanico de interfaces, incluyendo un chatbot llamado HABot para controlar la suite domótica con un lenguaje natural escrito (Esta misma función ha sido objeto de proyecto en un reciente

trabajo de fin de máster de la facultad de informática [Bru18]). Posee una API restful que permite una interoperatividad con servicios externos o desarrollos propios y soporte para todas las principales plataformas móviles (Android, IOs, Windows) ya que está programado en Java. No se limita tampoco en interoperatividad con buena parte de servicios y stacks de desarrollo existentes, y dispone de una extensa gama de dispositivos compatibles mediante Add-ons, como, por ejemplo, Chromecast o Philips Hue, otras soluciones domóticas como 'Max! Home Solution', o protocolos como MQTT o estándares de redes con TCP, UDP o WOL (Wake on LAN) entre otros. De hecho, salvando que su licenciamiento se limita a ser open source, es de lejos, una de las mejores opciones disponibles que cumplen con lo necesario para crear una solución domótica aislada. Al no generar dependencias de servicios externos, puede ser operada localmente o de forma remota y dispone de capacidad suficiente en cuanto a soluciones modulares para cubrir los futuros casos de uso que se plantearán. `Open Hab` está en sintonía de los objetivos de este documento, cubre las necesidades de nuestro alcance y mucho más. De hecho, cubre tantas posibilidades y entregan tantos servicios listos para ejecutar, que en las primeras pruebas nos sentimos totalmente vacíos. Perdimos la capacidad de experimentar los distintos niveles que conforman una suite domótica, por que este framework ya lo proporcionaba prácticamente todo. Además, su documentación es bastante extensa, quedándose el análisis de otras opciones fuera del tiempo establecido de investigación del estado del arte. Finalmente se decidió no implementarlo.

`MainFlux`: Se define como una plataforma tecnológica de código abierto y patente libre con licencia Apache 2.0. Incluso ofrece un dispositivo gateway para soluciones industriales y de computación desarrollada por la misma `Linux Foundation`. Esta planteado como Plataforma como servicio (PaaS) para integrar aplicaciones que interactúan con los nodos mediante un gateway. La plataforma, dispone de una versión gratuita para ser alojado localmente en un Docker o un equipo con SO Linux. Su documentación clarifica que el desarrollo de aplicaciones que conectaran a Mainflux se realizarán a través de protocolos bridging (como HTTP, MQTT,WebSocket o CoAP) [DRA19].

La opción de trabajar sin framework, aunque supone un mayor esfuerzo, puede despejar ideas preconcebidas que se plantean en las diversas soluciones, dando la opción de experimentar más abiertamente, sin las restricciones tecnológicas impuestas por las aproximaciones de dichos framework. En este punto de la documentación los autores decidieron unificar su propio conjunto de servicios, protocolos y desarrollo para crear una solución nueva, enfrentando así los retos que deben resolverse en su totalidad para alcanzar un despliegue de IoT enfocado a domótica con la estrategia de gateway y nodos.

2.3. Estándares y protocolos de comunicación y Stack de servicios

Orientando la selección de los servicios, estándares y protocolos necesarios para la creación de un prototipo, que permita alcanzar los objetivos planteados para este proyecto, surge la necesidad de plantear la estrategia de comunicación de dispositivos. Esto implica seleccionar un formato de depliegue de dispositivos en la red que comunica los nodos con el gateway.

2.3.1. Stack de servicios

Para nuestra estrategia de selección de servicios, se empieza determinando que BBDD que cimentará el resto del stack. Toda aplicación informática se sustenta en primera instancia sobre el almacenamiento de datos de manera persistente. Se segmentan en 2 categorías principales según como se relacionen entre sí dichos datos, esto es: BBDD relacionales, o no relacionales. Además de esta segmentación principal, existen otras características importantes en cuanto a la escalabilidad y diseño de cada modelo. Si consideramos cómo entendemos humanamente los datos necesarios para una suite domótica, una de las primeras impresiones, es que no sabemos con certeza cuántos dispositivos de sensorización o actuadores serán necesarios para cada caso de uso. De hecho, es muy probable que un mismo caso de

uso disponga de diferentes combinaciones de dispositivos para alcanzar una solución. Es, sin embargo, bastante obvio que básicamente almacenaremos pocos conceptos atómicos, los principales son: Ubicaciones, dispositivos en las ubicaciones y medidas y/o acciones de los dispositivos.

En la mayoría de proyectos de IoT observados en nuestras indagaciones, aparece en escena MongoDB. Es una elección popular gracias a su escalabilidad y flexibilidad, esto permite sortear más fácilmente la problemática de adaptarse y anticiparse a los cambios tan continuos que sufre el escenario del glsiot, ya que aparecen nuevos sensores continuamente, que generan nuevas muestras de datos, con funcionalidades distintas, etc. En una BBDD relacional es difícil realizar iteraciones del modelo de datos de esta manera. Por otro lado, el proceso de analizar unos datos que evolucionan continuamente, no sólo en contenido, sino en forma, hacen que su extracción y procesado se vuelva muy compleja. Las BBDD no relacionales son más amigables a la hora de definir criterios para extraer informaciones concisas dentro de un documento heterogéneo. Además, MongoDB está pensado para trabajar estrechamente con datos presentados en formato JavaScript Object Notation (JSON). Esto hace su uso muy conveniente en un entorno de aplicación que se base en estas estructuras de datos como ocurre con los motores de JavaScript (de los navegadores o de servidores) o la mayoría de aplicaciones móviles.

Destacar este último aspecto es determinante a la hora de seleccionar el entorno de ejecución que usará el servidor de la aplicación de domótica. En este aspecto, existen múltiples estrategias, todas ellas en una primera aproximación válidas. Se puede disponer de un servidor web como Apache o Nginx, programados en lenguaje PHP. Este tipo de servidor es capaz de establecer conexiones por el protocolo HTTP/HTTPS con un cliente solicitante, procesar una determinada información y entregar una respuesta codificada en texto plano para que el software del cliente lo reciba. También puede ser extendido para procesar comunicación

mediante MQTT, la cual es una excelente alternativa a HTTP/HTTPS dependiendo del contexto, y que analizaremos posteriormente.

Otra alternativa podría ser NodeJS. El creador de NodeJS tuvo la idea de coger el motor V8 de JavaScript del navegador Google Chrome y montarlo como el núcleo de NodeJS. NodeJS se programa en JavaScript, y representa un cambio de paradigma en la ejecución, puesto que sólo utiliza un hilo de ejecución y procesamiento de entrada y salida asíncronos; lo que evita bloqueos en el procesamiento y ayuda a optimizar los recursos del servidor. Además, es Open Source, multiplataforma y muy modular, lo cual atrajo y sigue atrayendo a una enorme cantidad de desarrolladores que contribuyen a su crecimiento añadiendo librerías en NPM (Node Package Manager), de forma que hoy en día, NodeJS posee infinidad de utilidades ampliamente testadas a su alcance, fácilmente reutilizables en cualquier proyecto.

Para el fin que ocupa al proyecto, es útil añadir el framework de ExpressJS, ligero y flexible orientado a la creación y exposición de una API REST con el objetivo de atender las peticiones de aplicaciones web y móviles. Esta adición sería casi imprescindible, pues la API básica de utilidades HTTP de NodeJS es muy poco amigable. Sin embargo, juntos, levantar un servidor web y una API REST completa y segura es una tarea asequible con poco más de un centenar de lineas de código. Otras librerías muy útiles y que nos conciernen podrían ser las que permiten integración con servicios como los arriba mencionados MQTT y MongoDB (para la cual está excepcionalmente bien orientado, dada la flexibilidad de JS y su natural compatibilidad con las estructuras de datos JSON).

2.3.2. Protocolos de comunicación

Mosquitto es un mediador de mensajes que incluye el protocolo MQTT. Además es de código abierto su documentación es extensa y detalla lo que supone una ventaja para los

desarrolladores.Es un buen candidato para ser usado en el desarrollo y pruebas de este proyecto.

Para discutir sobre las alternativas de comunicación del servidor con los dispositivos, hablaremos del arriba mencionado MQTT. Frente a la simple pero efectiva estrategia de HTTP/HTTPS, en proyectos de glsiot, hay un protocolo originalmente ideado por la compañía de IBM de licencia libre enfocado a una conectividad maquina a maquina (M2M), el código de dicho protocolo fue donado en 2011 al proyecto de Eclipse. Mientras que el protocolo HTTP/HTTPS puede manejar volúmenes grandes de información en su transferencia por la red de comunicaciones, el protocolo MQTT requiere menos ancho de banda, está orientado a proyectos con un bajo consumo y que dispongan de poco recursos de procesamiento; lo cual encaja con el enfoque necesario de una suite domótica, donde por ejemplo, si hay que obtener la temperatura de una habitación, todo se reduce a un único dato, un número (posiblemente decimal), o al menos, eso es lo que un usuario espera recibir. Esto implica que, un sensor de temperatura, en algún lugar de la casa, recibe una petición en la red en la que se encuentra de otro dispositivo, toma la medida y la envía a dicho dispositivo.

Si usamos el planteamiento de un servicio web, el sensor dispondrá de la capacidad de recibir una solicitud en un puerto abierto, cuando reciba la petición adecuada, responderá incluyendo en el cuerpo de la respuesta dicha medida y será procesada en el dispositivo que creó la petición. Esto significa que cada sensor/actuador será por sí mismo un servidor web con un puerto abierto que atiende peticiones de clientes. Si todos los dispositivos están conectados en la misma red, salvo que se apliquen restricciones específicas en el firewall, serán visibles entre sí, pudiendo encadenar llamadas y respuestas entre ellos, o a través de un dispositivo central que se encargue de hacer las solicitudes y cuyas respuestas recibidas sean procesadas para mostrarse en la aplicación móvil.

Desde el planteamiento del protocolo MQTT, la estrategia gira en torno al concepto de

suscripción de *topics* o temas, donde los dispositivos publican información de distintos temas y los suscriptores a dichos temas reciben la información cuando es publicada. A nivel físico, todos los clientes se conectan a un punto central, lo que define una topología de estrella (con sus ventajas e inconvenientes). Dicho nodo central es el servidor definido como *broker*, y es el único que sabe a qué topic está suscrito cada cliente. Esto significa que cada uno de los sensores/actuadores de la red ignora al resto de los dispositivos, ya que sólo atienden la comunicación con el broker a través de una conexión glstcp permanente. Esta estrategia es diferente a la usada en el protocolo HTTP, ya que no es necesario hacer una petición para recibir información de un cliente. Simplemente el broker da o solicita información en los topics a los que los dispositivos están suscritos. Para proyectos de este tipo, el hecho de que los dispositivos no se relacionen entre sí, dan lugar a una gran ventaja, la escalabilidad. en el año 2014 se ha convertido en un estándar OASIS. MQTT soporta cifrado mediante SSL. Es un protocolo fiable. Eso se debe a que tiene implementado Quality of Service (QOS).

La escalabilidad de una suite domótica es un factor determinante al considerar su implementación. Que los dispositivos que conforman la red puedan cambiar con facilidad su organización, número o capacidades es un valor añadido.

2.3.3. Estándares en IoT

La domótica se remonta a los años 70, uno de los primeros hitos fue el protocolo de comunicaciones X-10 [Com] de automatización de dispositivos en la línea eléctrica de un hogar. Con él, se puede utilizar la propia red como canal de comunicaciones mediante ráfagas de pulsos. El ancho de banda de 256 dispositivos simultáneos es una cantidad más que suficiente para interactuar con los dispositivos de un hogar, si calculamos que cada elemento susceptible de ser automatizado como por ejemplo persianas, estufas, enchufes, e interruptores ocupase un espacio de este ancho de banda, seguirían sobrando espacios en un piso de 150 metros cuadrados.

En pleno 2019 pueden adquirirse los dispositivos necesarios para instalar una red X-10 que incluye interruptores, actuadores, sensores, transmisores, interfaces y unidades de control (todos ellos necesarios para obtener el control completo de la red) por precios con rangos entre 20 a 70 euros por cada elemento. Esto supone inversión elevada si se tiene en cuenta un hogar de varias habitaciones. Hay que considerar además ciertas limitaciones como que cada controlador es capaz de manejar un número limitado de dispositivos. En cuanto a las aplicaciones necesarias para gestionar el sistema, X-10 posee alternativas de código libre para su desarrollo como `Minerva`. También existen interfaces hardware que traducen el protocolo X-10 a APIs de servicios web como el dispositivo `ioBridge` que supuso una disrupción en el ámbito del IoT y la automatización del hogar. El problema del despliegue de soluciones basadas en X-10 no radica en su protocolo sino en los costes, que generalmente sobrepasan los cientos de euros para las configuraciones más sencillas. Además, esta instalación requiere de un proceso de obra, ya que es necesario empalmar los componente a la red eléctrica del hogar. En realidad, la opción de usar dispositivos del protocolo X-10 esta condicionada a que la red eléctrica del hogar se instale durante el proceso de edificación con vistas a utilizar este sistema. De otra forma, será necesario planificar obras y esto complica la facilidad de crear un prototipo asequible.

Una década después surgiría el Sistema de Cableado Estructurado (SCE) que permitía el trasporte de datos y voz, esto supuso la aparición del concepto Edificio Inteligente. Sin embargo, esto requiera de una instalación compleja difícilmente viable en edificios ya existentes, como es el caso que ocupa el alcance de este proyecto. Con la aparición de las redes inalámbricas, y sus posteriores estándares de comunicación como el Wireless Fidelity (WI-FI), se popularizaron los sistemas de domótica aplicada directamente sobre los dispositivos, sin tener en cuenta la red eléctrica que los hace funcionar.

A inicios del milenio se aprobó la especificación del estándar de Zigbee, basado en el

estándar IEEE 802.15.4 de redes inalámbricas de área personal (WPAM). Se destacan las características de bajo consumo, topología y fácil integración, que lo convierten en una de las opciones mas adecuadas para proyectos de IoT. Permite la creación de dispositivos con batería gracias a un consumo de apenas 30mA trasmitiendo datos inalámbricos y de apenas 3 microamperios en reposo. Poder desplegar una red WI-FI en malla posee un atractivo especial que puede ser una de las mejores estrategias en hogares con múltiples plantas, donde la fuerza de señal inalámbrica se disipa con rapidez al tener que atravesar los suelos y paredes. Su precio lo convierte en un producto muy competitivo, y puede adquirirse por menos de una docena de euros en tiendas digitales en internet. Otro aspecto particularmente atractivo es la seguridad, existen controles de acceso de los dispositivos con autenticación y cifrado de clave simétrica, así como comprobación de integridad de mensajes, estas técnicas reducen la posibilidad de que las comunicaciones entre dispositivos puedan verse comprometidas. Las pruebas de laboratorio realizadas por el INCIBE demuestran que una configuración adecuada hace de la red de dispositivos con estándar Zigbee una solución robusta [ndcdEMS].

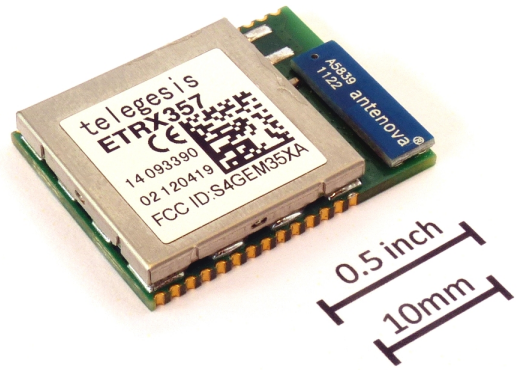

Figura 2.4: *Modulo transductor de Zigbee*[1]

Un ejemplo más de soluciones IoT es `6LoWPan` ofrece un eficiente uso del protocolo IPv6 sobre el estándar IEEE 802.15.4 a través de redes locales inalámbricas ideadas para actuar

como despliegues de red en malla [MKHC07]. Entre sus usos más comunes se encuentra la industria IoT, la agricultura inteligente y las smart home. Este protocolo dispone además de una capa de seguridad adicional basada en autenticación y cifrado AES-128. Esto permite implementar mecanismos adicionales como Transport Layer Security (TLS) o firma digital en las comunicaciones. Su uso se planifica en transferencia de pequeñas tasas de datos, pero como resultado reduce el consumo de los nodos de la red. La siguiente figura 2.5 extraída del articulo 'The analysis of 6LoWPAN technology' refleja un ejemplo de despliegue de esta tecnología [ML08].

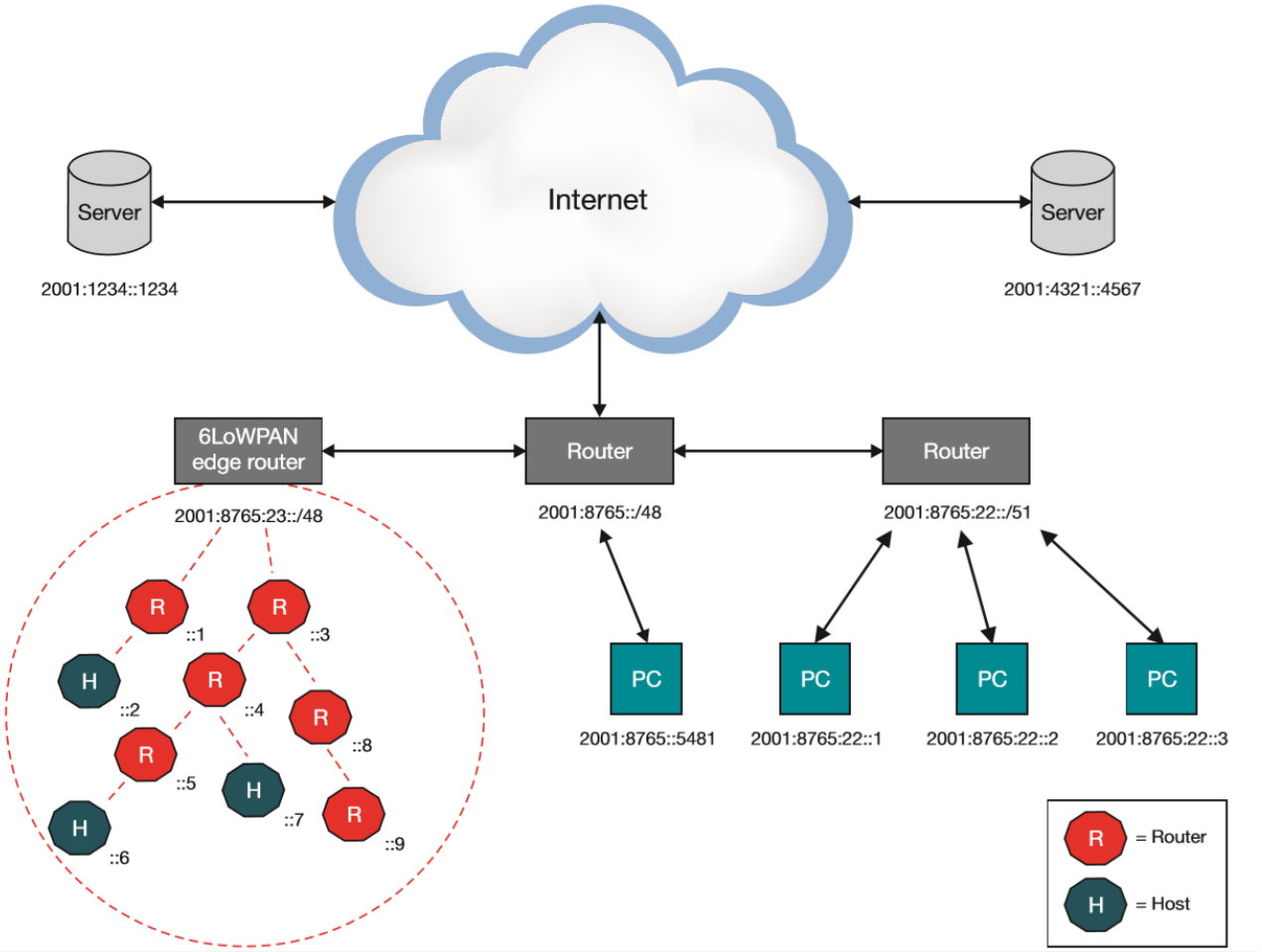

Figura 2.5: *Ejemplo de despliegue de una red 6LowPan con servicio en la nube*

Otra de las tecnologías en iot es `LoRa`, su base tecnológica está definida en el RFC8376 [SF19] es el nombre común de la tecnología 'Low Power Wide Area Network' (LPWAN). Los despliegues de estas redes inalámbricas son de tipo estrella, pero la comunicación entre nodos y gateway se establecen en canales individuales donde la velocidad de transferencia de datos se alcanza según la distancia entre dispositivos, la duración y consumo energético del mensaje a enviar. El efecto de esta técnica es que los canales con distintas velocidades de transferencia no interfieren entre si, lo cual genera un amplio abanico de canales simultáneos que pueden establecerse con el gateway.

2.4. Hardware disponible

Se han descrito algunas de las tecnologías características de las aplicaciones, topologías de red y protocolos de comunicación utilizados en domótica. Pero todo este entramado de software y señales deben operar en hardware físico. Más concretamente en múltiples dispositivos de hardware. En IoT el abanico de fabricantes de dispositivos orientados a sensorización y actuadores es tan extenso que simplemente está fuera de todo alcance el enumerarlos en este documento; aun así, considerando los objetivos definidos en el proyecto, que incluyen el uso de hardware open source, y costes de adquisición reducidos, podemos reducirlo a una lista de opciones acotada. Este ananlisis cubre el nodo central o gateway y los nodos sensores y actuadores.

2.5. Opciones Hardware para un gateway

Partimos del concepto de suite domótica basado en dispositivos conectados a un nodo principal o gateway, el cual actúa como router WI-FI. Dicho gateway posee un adaptador de red adicional que permite una conexión con la red de internet y que sea capaz de ejecutar servicios y aplicaciones actuando como servidor. Estos requisitos nos orientan a disponer de un ordenador completo, será necesaria la flexibilidad de un SO que nos permita experimen-

tar diferentes planteamientos, sin dejar de tener en cuenta que este ordenador tendrá una disponibilidad continua, y debe tener un consumo energético bajo y unos recursos suficientes para actuar como cimiento del prototipo.

Hace una década habría sido necesario apuntar a un equipo muy especializado y esto generalmente se traduce en un incremento del precio del equipo. Hoy, en cambio, disponemos de muchas opciones de ordenadores con un reducido factor de forma, bajo consumo eléctrico y recursos más que suficientes para cubrir muchos prototipos ligeros. Hablamos de la muy conocida Raspberry Pi y similares que han surgido con el tiempo (como Orange Pi, Banana Pi, Odroid o Matrix ARM [Noe15]), pueden encontrarse en la figura 2.6 algunos aspectos técnicos de sus capacidades comparadas entre sí.

Brand	Raspberry Pi	Raspberry Pi	Raspberry Pi	Banana Pi	Banana Pi	Orange Pi	Orange Pi	Orange Pi	Orange Pi	ODROID	ODROID	ODROID
Board	Zero	2 Model B	3 Model B	M2	M3	One	PC	Plus	Plus 2	C1+	C2	XU4
SoC Vendor	Broadcom	Broadcom	Broadcom	Allwinner	Allwinner	Allwinner	Allwinner	Allwinner	Allwinner	Amlogic	Amlogic	Samsung
SoC Chip	BCM2835	BCM2836	BCM2837	A31s	A83T	H3	H3	H3	H3	S805	S905	Exynos 5422
SoC Process	40nm	40nm	40nm	40nm	28nm	28nm	28nm	28nm	28nm	28nm	28nm	28nm
CPU Cores	1	4	4	4	8	4	4	4	4	4	4	4+4
CPU Design	ARM1176JZF-S	Cortex A7	Cortex A53	Cortex A7	Cortex A7	Cortex A7	Cortex A7	Cortex A7	Cortex A7	Cortex A5	Cortex A53	Cortex A15/A7
CPU Freq	1GHz	0.9GHz	1.2GHz	1GHz	1.8GHz	1.5GHz	1.5GHz	1.5GHz	1.5GHz	1.5GHz	2GHz	2.1GHz/1.5GHz
CPU Instruction	ARMv6	ARMv7	ARMv8	ARMv7	ARMv7	ARMv7	ARMv7	ARMv7	ARMv7	ARMv7	ARMv8	ARMv7
GPU Vendor	Broadcom	Broadcom	Broadcom	PowerVR	PowerVR	ARM	ARM	ARM	ARM	ARM	ARM	ARM
GPU Design	VideoCore IV	VideoCore IV	VideoCore IV	SGX544MP2	SGX544MP(1/2?)	Mali 400MP2	Mali 400MP2	Mali 400MP2	Mali 400MP2	2x Mali 450	3x Mali 450	Mali T628 MP6
GPU Freq	250MHz	250MHz	400MHz	355MHz	700MHz	600MHz	600MHz	600MHz	600MHz	600MHz	700Mhz	600MHz
H264 Dec	1080P30	1080P30	1080P30	1080P60	1080P60	1080P60	1080P60	1080P60	1080P60	1080P60	1080P60	1080P60
H264 Enc	1080P30	1080P30	1080P30	1080P30	1080P60	1080P30	1080P30	1080P30	1080P30	1080P30	1080P60	1080P60
H265 Dec	None	None	None	None	1080P30	4KP30	4KP30	4KP30	4KP30	1080P60	4K60	None
Memory	512MB DDR2	1GB DDR2	1GB DDR2	1GB DDR3	2GB DDR3	512MB DDR3	1GB DDR3	1GB DDR3	2GB DDR3	1GB DDR3	2GB DDR3	2GB DDR3
Memory Freq	400MHz	400MHz	400MHz	432MHz	672MHz	672MHz	672MHz	672MHz	672MHz	792MHz	912MHz	750MHz
Storage	MicroSD	MicroSD	MicroSD	MicroSD	MicroSD/USB SATA 2.0	MicroSD	MicroSD	MicroSD/USB SATA 2.0	MicroSD/USB SATA 2.0	MicroSD/eMMC	MicroSD/eMMC	MicroSD/eMMC
Storage Onboard	None	None	None	None	8GB eMMC	None	None	8GB eMMC	16GB eMMC	None	None	None
Storage	SD2.0	SD2.0	SD2.0	SD3.0	SD2.0/eMMC 4.4	SD3.0	SD3.0	SD3.0/eMMC 4.5	SD3.0/eMMC 4.5	SD3.0/eMMC 4.5	SD3.0/eMMC 5.0	SD3.0/eMMC 5.0
USB 2.0	1 OTG	4	4	4 + 1 OTG	2 + 1 OTG	1 + 1 OTG	3 + 1 OTG	4 + 1 OTG	4 + 1 OTG	4 + 1 OTG	4 + 1 OTG	1
USB 3.0	0	0	0	0	0	0	0	0	0	0	0	2
Ethernet	None	100Mb	100Mb	1Gb RTL8211E	1Gb RTL8211E	100Mb	100Mb	1Gb	1Gb	1Gb	1Gb RTL8211F	1Gb
Wireless	None	None	802.11N BCM43438	802.11N AP6181	802.11N AP6212	None	None	802.11N RTL8189ETV	802.11N RTL8189ETV	None	None	None
Bluetooth	None	None	Bluetooth 4.1	None	Bluetooth 4.0 AP6212	None	None	None	None	None	None	None
IrDA	None	None	None	Yes	Yes	None	Yes	Yes	Yes	Yes	Yes	None
HDMI	1200P60	1200P60	1200P60	1200P60	1200P60	4KP30	4KP30	4KP30	4KP30	1200P60	4K60	1600P60
RTC	No	No	No	No	No	No	No	No	No	Yes	No	Yes
Power Req	5V1A	5V2A	5V2.5A	5V2A	5V2A	5V2A	5V2A	5V2A	5V2A	5V2A	5V2A	5V4A
Power Plug uUSB	Yes	Yes	Yes	No	Some Boards	No	No	No	No	Yes	Yes	No
Power Plug Barrel	No	No	No	4/1.7mm	4/1.7mm Some Boards	4/1.7mm	4/1.7mm	4/1.7mm	4/1.7mm	2.5/0.8mm	2.5/0.8mm	5.5/2.1mm Barrel
Android	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Linux Mainline	Yes	Yes	Yes	Yes	4.6	Yes	Yes	Yes	Yes	No	No	Yes

Figura 2.6: *Comparativa de características técnicas de microordenadores*

Para el caso que nos ocupa necesitamos que disponga de al menos dos adaptadores de red, uno de ellos inalámbrico, aunque puede subsanarse la falta del mismo mediante adaptadores inalámbricos Universal Serial Bus (USB) (una estrategia muy común en los primeros modelos de Raspberry Pi hasta la serie 3). No es necesario que disponga de un procesador gráfico ya que operaremos de forma remota el ordenador mediante conexiones de Secure SHell (SSH).

Será un aspecto muy positivo que disponga de una interfaz para conectar dispositivos USB. Esto último responde a la posible necesidad de conectar placas microcontroladoras para su programación.

Raspberry Pi es posiblemente el uno de los mejores ejemplos de hardware abierto disponibles en el mercado. El diseño de su circuito impreso puede descargarse libremente para crear versiones modificadas. [Foub]. Su precio puede variar notablemente dependiendo del vendedor, pero generalmente no debe superar los 35 euros. Dispone de capacidad suficiente para ejecutar distintos SO basados en arquitectura ARM. En el capítulo siguiente de la propuesta se evaluará que SO utilizar. De entre las distintas opciones disponibles, Raspberry Pi posee una de las comunidades de usuarios más grandes del mundo, posee una amigable documentación de uso e interminables ejemplos de uso y proyectos disponibles en la red. Sus especificaciones técnicas son suficientes para sostener los servicios necesarios para un prototipo de suite domótica, y puede alimentarse con una toma de USB de 5 Voltios. Uno de los modelos mas vendidos es la 3B+, ilustrado en la siguiente figura 2.7.

Figura 2.7: *Ordenador Raspberry Pi 3B*

Esto cubre el soporte físico necesario para crear un gateway. Pero por sí solo no es su-

ficiente. Los dispositivos que conectaremos a esta solución pueden presentarse en distintas opciones de hardware que pueden operar entre sí y con el propio gateway mediante conectividad WI-FI. Gracias a los protocolos de comunicación disponibles, un gateway puede comunicarse con dispositivos de distintos fabricantes, presentando en la aplicación a todos ellos como dispositivos homogéneos con diferentes funciones.

2.6. Sensores y actuadores

En esencia, la clasificación de actores que conforman la red domótica se dividen en el gateway (aunque pueden existir varios),los nodos, y los sensores y actuadores. Esta clasificación responde a la facilidad de entender rápidamente si un dispositivo actúa como entrada o salida del sistema. En IoT es habitual disponer de hardware especializado de bajo consumo y orientado a protocolo concretos de comunicación para ser usados en framework, como ocurre con OpenHab, que facilita bastante la tarea de incluir nuevos dispositivos a la suite domótica si el producto a integrar ya está implementado. Si se desea disponer de la flexibilidad de cambiar radicalmente el comportamiento de un actor, modificando su comportamiento y hardware, es mejor plantear el uso de placas microcontroladoras programables.

Uno de los ejemplos más conocidos y amigables de usar en placas microcontroladoras programables es Arduino. Estas placas disponen de conexiones Imput/Output (I/O) que pueden usarse para ampliar su abanico e capacidades técnicas. En prototipos de IoT, es habitual dotar a una placa con conectividad WI-FI mediante un adaptador inalámbrico conocido como shield.

En los últimos años ha aparecido un pequeño microcontrolador con chip de WI-FI conocido generalmente como modulo ESP-01, tal y como se muestra en la figura 2.8 este modulo es muy compacto. El chip permite una comunicación con el protocolo TCP/IP en una red

inalámbrica. Se clasifica como hardware RF/IF y RFID CI de transceptor RF, capaz de operar con el protocolo 802.11b/g/n de 2.4GHz. El modelo ESP8266 que actualmente se comercializa posee capacidades superiores a su predecesora ESP8265 incluye una mayor capacidad de memoria flash interna, suficiente para abordar la mayoría de dispositivos IoT que pretendan actuar como actuadores o sensores. Sin embargo, trabajar y programar estos chips puede resultar engorroso si no se dispone de glsusb con interfaz para escritura en serie, también puede hacerse con una placa de Arduino separando la cucaracha del microcontrolador del Arduino y cableando la conexión necesaria para programarse desde un ordenador.

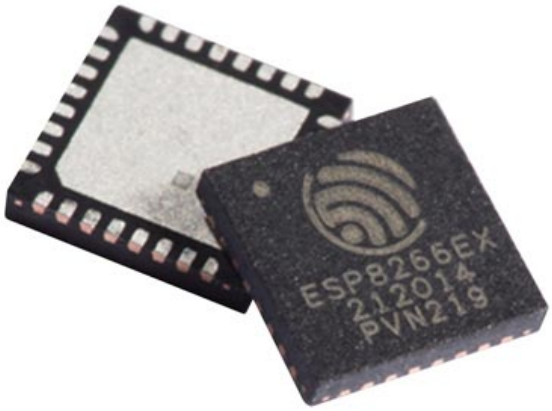

Figura 2.8: *Controladora ESP8266*

Este microcontrolador opera con apenas 3.3V DC, lo cual es considerado como una ventaja a la hora de incluir este modulo en proyectos que operan sobre otras placas microcontroladoras como Arduino. Sin embargo, esta opción es válida sólo para la creación de prototipos funcionales, ya que el modulo ESP-01 requiere de un amperaje de al menos 200mAh para operar con normalidad. Los pines de alimentación de 3,3V de placas como Arduino ofrecen unos 50mAh de intensidad de corriente. Esto es suficiente para que el modulo opere, pero ocasionalmente puede causar daños en el modulo y generalmente el funcionamiento de la conexión es de baja calidad, con poca intensidad de señal, pérdidas de paquetes y caídas de la conexión. La estrategia más extendida en facilitar una conexión estable de

alimentación al modulo separado de la alimentación recibida por la placa microcontroladora en la que opera.

También pueden ejecutarse construcciones simples en una breadboard para un divisor de potencia mediante resistencias que transforme la señal de 5V de las placas microcontroladores, que habitualmente ofrecen un mayor valor de amperios, para dar una señal de 3.3VDC con mas de 200mA. Siendo realistas, no es ni siquiera necesario crearlos si se considera las opciones de venta de algunos fabricantes que montan una microcontrolador con chip ESP8266 en una ensamblada con las resistencias necesarias, interruptores y habitualmente sensores o actuadores como el mostrado en la figura 2.9 a modo de módulo compacto alimentado por pines a 5V. El precio de estos módulos apenas alcanza un par de euros si se solicita a vendedores chinos, pero, es conveniente entender los riesgos que se asume al adquirir estos dispositivos de fabricantes clónicos con dudosos controles de calidad, que pueden seleccionar componentes incompatibles son redes WI-FI como el suministrado por el adaptador inalámbrico de una Raspberry Pi 3B. Más información sobre estos problemas están ampliados en anexo B de troubleshooting.

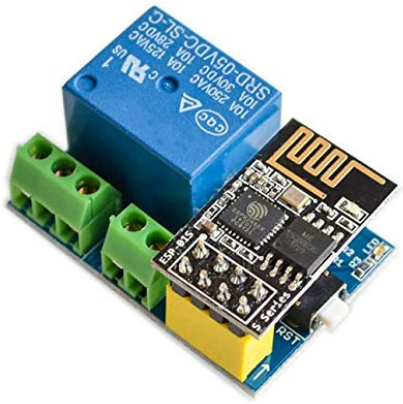

Figura 2.9: *ESP8266 5V Modulo Rele*

Con las complicaciones de alimentación del modulo `ESP8266` surge al poco tiempo su sucesor natural, las placas nodeMCU. Conocidas por ser plataformas de código abierto para IoT más versátiles y accesibles en coste de adquisición. Uno de los aspectos que popularizó estos dispositivos fue la posterior portabilidad de la biblioteca de MQTT, permitiendo al LUA de la IPv6 over Low power Wireless Personal Area Networks (SoC) desplegar dicho protocolo. Aunque el proyecto de mantenimiento de firmware fue abandonado en 2015 por los autores originales, la comunidad de usuarios siguió mejorando el código hasta fecha de hoy, convirtiendo el nodeMCU en uno de los dispositivos más demandados en proyectos, en la figura 2.10 puede observarse uno de los muchos modelos derivados. Gracias a su bajo coste de precio, inferior a una decena de euros por placa, que dispone de 12 pines de General Purpose Input/Output (GPIO) permitiendo implementaciones complejas que aprovechan la conectividad inalámbricas para proyectos de IoT.

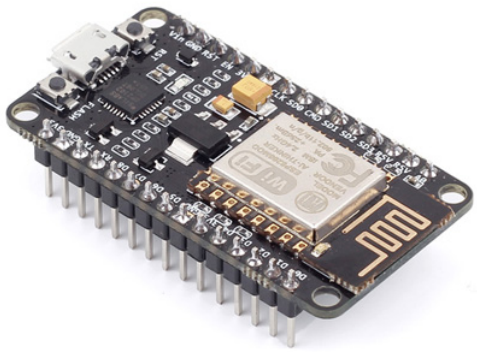

Figura 2.10: *Microcontroladora NodeMCU*

Capítulo 3

Requisitos

El objetivo del pryecto es crear una solución para domótica basada en ser libre. En esta solución se consideran 2 perfiles de usuarios. Por un lado, un perfil de `usuario` sin conocimientos específicos de informática. Su necesidad de domótica se centra en monitorizar aspectos ambientales del hogar, o manipular mediante un móvil las luces, pudiendo verificar si las luces están encendidas en una sala sin necesidad de ir a la misma. El segundo perfil, el de `administrador` posee conocimientos de programación en lenguajes de alto y bajo nivel y gestión de un SO Linux. Será el encargado de planificar, desplegar y gestionar la suite domótica en base a las especificaciones impuestas por el usuario.

Como restricción, se busca que el coste de la solución domótica que se aplique, no debe exceder un presupuesto por encima de 100 euros en componentes. Esta restricción, además, debe considerar que la escalabilidad de la solución tiene un alcance indefinido, pudiendo solicitarse la agregación a la red domótica de cualquier clase de sensor y actuador en el futuro. Sin embargo, sí está definida la intención del usuario de monitorizar la temperatura de una estancia y su control de luces. Se obvian los costes de desarrollo, ya que en este supuesto escenario, el administrador ya ha decidido la creación de un despliegue de IoT propio, el cual está reflejado en esta documentación. La estancia en la que se desarrolla el escenario es una habitación subterránea con un único acceso y sin ventanas. Dispone de múltiples puntos de conexión a la red eléctrica. El principal problema del usuario en

la sala, es el interruptor de luz que no se encuentra junto a la puerta de entrada, si no bajando las escaleras. También preocupa la temperatura y humedad de la estancia en la cual se encuentran dos cubas caseras para fermentar vino. Si la temperatura no es estable, esto afectará a la calidad final del vino. Pero antes de determinar si en necesario instalar equipamiento deshumidificador, o sistemas de climatización, hay que conocer la situación actual. El administrador expone el despliegue IoT que se aplicará en la estancia mostrada en la figura 3.1. Esta suite debe cubrir la necesidad de controlar el estado de una bodega de uso personal en una casa, pudiendo obtener valores ambientales de la estancia y controlar el sistema de iluminación de la estancia que accede a la misma, pudiendo alternar el estado de las luces de manera automática o manualmente, desde la aplicación.

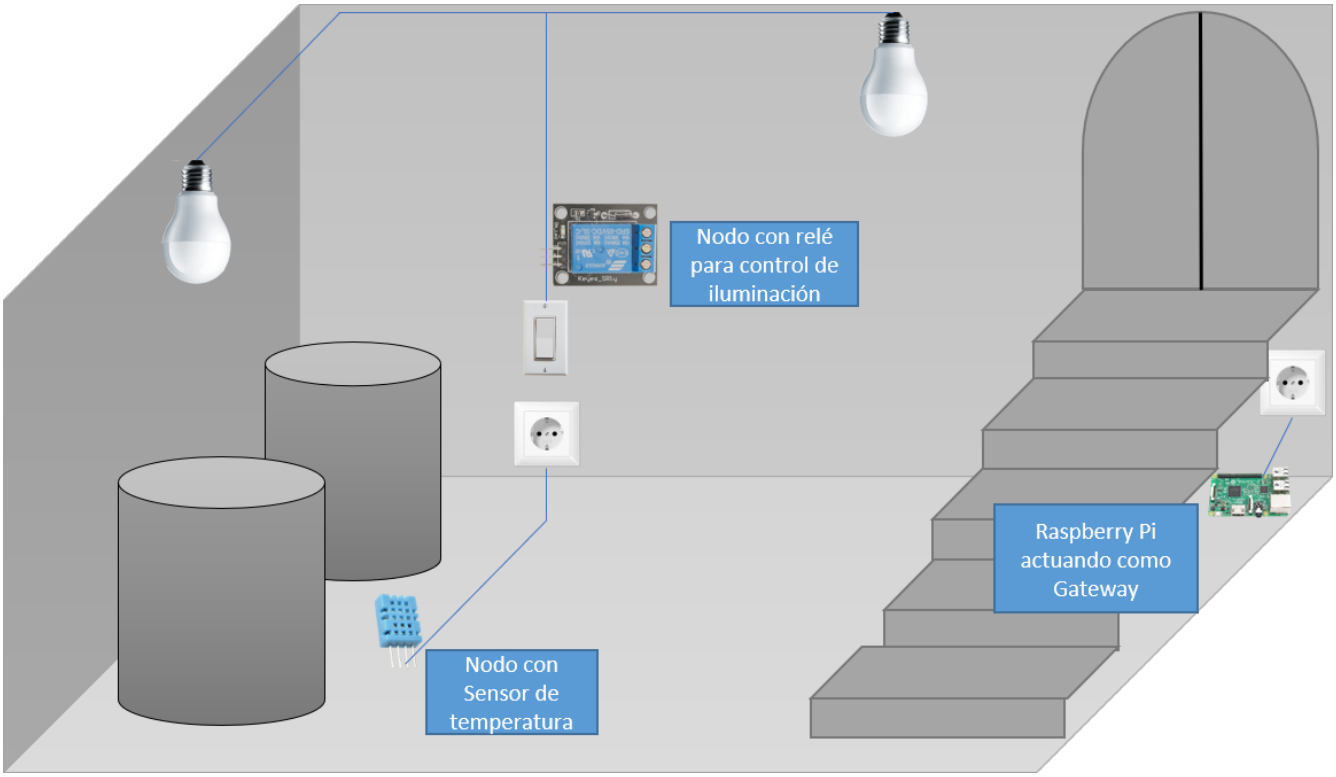

Figura 3.1: *Escenario propuesto para el despliegue en una estancia*

Para facilitar el uso por parte del usuario de esta suite domótica, se instalará en su móvil

una aplicación que permitirá gestionar la configuración del sistema, consultando los registros de los nodos sensores, los estados de los nodos actuadores, manipular dichos estados, y agregar nuevos nodos en el despliegue de una instancia, o generando instancias separadas para organizar con más facilidad. Todo este abanico de casos de uso serán explicados en el capitulo 5. Para flexibilizar su uso a distancia es necesario que la aplicación gestione los dispositivos tanto desde dentro de la casa, como desde fuera. Eso implica que el administrador configure una redirección en el router domestico del hogar hacia un servidor ejecutado en un ordenador de la casa.

Aunque inicialmente el prototipo de solución domótica puede registrar temperaturas y humedad de una instancia así como alterar el estado de un relé que controle el estado de las luces, será necesario planificar un método que permita al administrador agregar nuevos tipos de sensores y actuadores a las opciones disponibles. Eso implica crear programaciones específicas para cada tipo de dispositivo, y también la posibilidad de que cada nodo pueda ejecutarse sobre diferentes placas microcontroladoras, en función de las necesidad que puedan surgir.

La suite domótica sera ejecutada en un ordenador de bajo coste con capacidad inalámbrica, que permite la conexión de nodos inalámbricos con roles de actuador o sensor. Puede operar desde dentro de la red local en la que operan los nodos, o desde fuera de la misma con conexiones remotas, mediante una aplicación para dispositivo móvil con sistema operativo Android. Si bien, la operatividad remota es opcional, se debe alcanzar la gestión de la suite en el entorno de la red local, cumpliendo así con el objetivo de disponer de un sistema aislado y autónomo, que no dependa de servicios externos para su funcionamiento. Una vez alcanzada una implementación funcional del prototipo se aplicará en dos casos de usos que ejemplifican la entrada y salidas características de todo sistema de domótica: la recepción, proceso y presentación de datos de un dispositivo sensor en la aplicación móvil; y la gestión

de un actuador desde dicha aplicación.

3.1. Despliegue de la Solución

Para lograr la propuesta, será necesario considerar de entre las opciones estudiadas en el Capítulo 2 aquellas que se ajustan mejor al alcance de esta propuesta. Dadas las distintas capas de hardware, arquitectura de red y software que componen este proyecto, la propuesta será dividida en tres conceptos modulares, permitiendo un desarrollo individual y en paralelo respecto de cada uno.

- Se configurará un gateway, que actúe como router de la red de dispositivos de la suite domótica y servidor de la aplicación. Dispondrá de la capacidad de ser operado de forma remota o local, almacenará los servicios y aplicaciones necesarios para que funcione la suite domótica ejecutándose en un ordenador de bajo consumo.
- Se generarán distintos dispositivos sensores o actuadores que podrán incluirse en la red del gateway y podrán ser operados desde una aplicación móvil.
- Se diseñará y creará la aplicación móvil (frontend) y el servidor (backend) que darán al usuario la capacidad de gestionar la suite domótica.

Se tratará de alcanzar una cierta descentralización de los dispositivos y la propia Raspberry, basándose en el concepto de nodo principal (gateway), que habitualmente se observa en las plataformas de pago. Dichos planteamiento se basan en que todo sensor/actuador que forme parte de red de dispositivos de una solución de domótica actual, es gestionada a través de un nodo. En vez de conectar los dispositivos inalámbricos al router de la casa, se conectan al nodo y éste, a su vez, es quien se conecta a la red local del hogar, para así conectarse con los servicios externos. A la hora de conectar nuevos dispositivos a la red del nodo, las distintas plataformas abordan el problema de la misma manera. El usuario

final compra un nuevo dispositivo, lo enciende, dejándolo en un estado de ïnclusión.ª la red domótica, después, desde la aplicación de móvil, se indica al nodo que se quiere añadir un nuevo dispositivo, y tras seguir las indicaciones, el dispositivo se registra en la red del nodo. Esto, sin embargo, tiene algunos inconvenientes en el proceso de ïnclusión", y aunque la probabilidad es baja, puede suceder que dos nodos de distintas viviendas, que están registrando dispositivos simultáneamente, terminasen, registrando un dispositivo que no les corresponde. Esto es una vulnerabilidad de seguridad grave y una vertiente adicional que incluir en las motivaciones del estudio y desarrollo de una suite domótica libre que permita nuevas formulas de funcionamiento. Se ha planteado este problema, junto con las 3 motivaciones principales, para crear un proceso de ïnclusión"de dispositivos al nodo, que parta de una conexión por cable (vía USB) y resuelva este inconveniente, simplificando el proceso de las soluciones privadas, que en ocasiones pueden fallar.

3.2. Propuesta de casos de uso

Para el perfil de usuario se enumeran las siguientes acciones que deben poder cumplirse para satisfacer los requisitos demandados en la estancia.

1. Proceso de vinculación de un dispositivo a una habitación, a partir de un asistente que el usuario debe seguir en su aplicación móvil con este fin, y que tras finalizar el proceso, dará lugar a un dispositivo operativo y en funcionamiento.

2. Interacción del usuario con la suite domótica para consultar la temperatura y/o humedad de una estancia. Para ello, es necesario disponer de un dispositivo inalámbrico con un sensor que recoja las mediciones y puedan ser mostradas al usuario en la aplicación movil mediante su smartphone.

3. Gestión por parte del usuario de un actuador basado en un interruptor de corriente, pudiendo consultar su estado actual y alternar dicho estado, también desde un smartphone.

Para el perfil de administrador será necesario disponer de procedimientos que permita incluir el soporte de nuevos tipos de sensores y actuadores entre las opciones disponibles que puede desplegar el usuario. El proceso de incluir físicamente nuevos nodos en la red debe ser simple, hasta el punto de que el usuario pueda hacerlo por si mismo sin supervisión del administrador.

1. Integración de múltiples opciones de hardware compatibles con la suite domótica.

2. Vinculación de dispositivos con la red de suite domótica mediante USB, en lugar del clásico emparejamiento WI-FI.

Capítulo 4

Arquitectura e Implementación

Existen múltiples niveles de implementación del prototipo planteado en este documento. El nivel más básico e imprescindible implica un ordenador con SO Linux que actuará como controlador de la infraestructura de dispositivos que dan forma al sistema en su conjunto, esto será referido en adelante como gateway.

4.1. Despliegue IoT de la solución domótica

Para disponer de una base tecnológica sólida, pero fácil de adquirir, sobre la que experimentar y entender la complejidad de una suite domótica y sus planteamientos, en este proyecto se opta por usar hardware amigable como `Raspberry Pi` para el gateway y placas microcontroladoras `nodeMCU` y los comúnmente llamados `modulos ESP8266`, que disponen de una extensa comunidad de usuarios y documentación. En esa misma linea se buscará software libre. No sólo se puede aprovechar gran parte del trabajo ya creado en programación de script para dispositivos como sensores y actuadores, aparte de ofrecer libertad para el usuario final de adaptar y modificar a su criterio tanto el software como el hardware. El gateway actúa como concentrador de las conexiones de nodos, y dispone de los servicios necesarios para actuar como `broker` de información para el servidor `Node.js` que será el núcleo del sistema, facilitando conexiones a clientes, los cuales mediante una APP móvil

podrán conectarse para operar la suite domótica. En la siguiente figura 4.1 se observa cómo la aplicación móvil que se desarrollará puede conectarse a través de la red de internet o de manera directa al router domestico, que conectará a su vez con el gateway a través de una conexión de cable. El gateway establece así el limite externo de la red de este despliegue de IoT y es a la vez el proveedor de servicios para la aplicación móvil, eliminando servicios de aplicación de terceros alojados en la red de internet como requisito innecesario para que pueda operar.

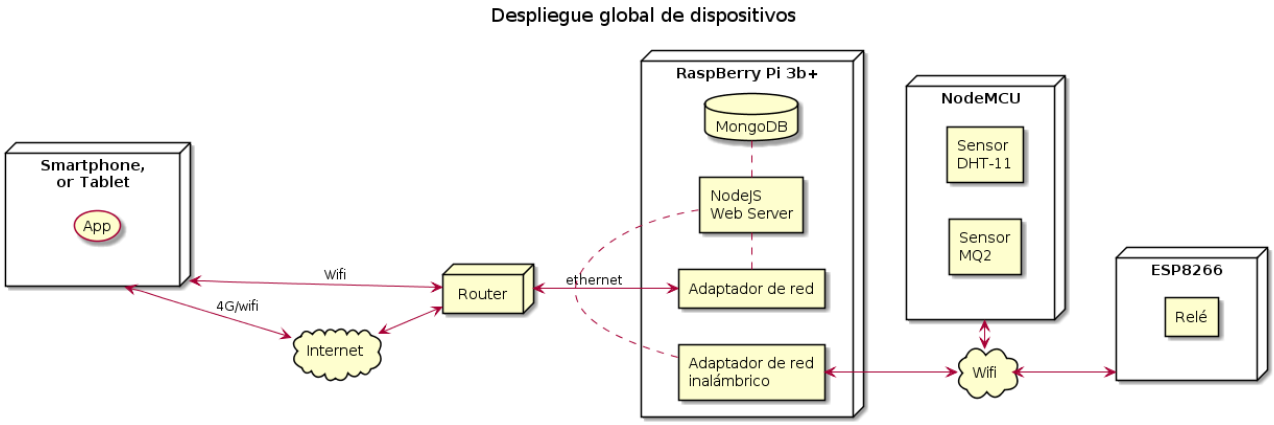

Figura 4.1: *Diagrama de despliegue global de dispositivos*

4.2. Definición del gateway

El gateway representa el núcleo de nuestra suite domótica. Al no existir una dependencia de procesos externos y computación en la red de internet, este elemento representa el dispositivo en la posición más elevada de la jerarquía de la suite domótica. Los nodos dispuestos en un hogar lo usan como concentrador de comunicaciones para que el sistema opere, y la aplicación móvil lo usa como servidor para obtener la información registrada por esos dispositivos e interactuar con ellos. Si bien los nodos son capaces de registrar datos por su cuenta, estos deben ser entregados en el gateway para ser procesados de forma útil para la aplicación.

El gateway ejecuta `Raspbian`. Un SO `GNU/Linux` diseñado específicamente para la arquitectura ARMv7 de los distintos modelos de ordenadores Raspberry. Concretamente, en este proyecto se utiliza la serie de distribuciones `Lite`, orientadas a la ejecución del sistema operativa mediante interfaz de linea de comandos en terminal. Esta decisión está fundamentada en desprenderse de la necesidad de periféricos externos e interfaces de `I/O` analógicas para controlar el SO. Toda configuración del sistema será realizada mediante conexiones por SSH. El proceso de instalación y configuración de conexiones está ampliado en el apéndice A.2 del documento. La selección de esta distribución en lugar de un SO estrictamente orientado a IoT, o para sistemas empotrados, se debe a la flexibilidad de aplicaciones y servicios que pueden instalarse durante los periodos de prueba y diseño.

Una vez establecidos los canales de comunicación, se instalarán aplicaciones que permitirán ejecutar los servicios necesarios para actuar como gateway. La estrategia de comunicación entre los distintos dispositivos que conforman la suite domótica se basa en un servidor `NodeJs` que establece conexiones vía WI-FI de 2.4Gh con los nodos mediante el protocolo MQTT. El servidor a su vez establece conexión con la aplicación móvil vía WI-FI o internet, mediante el protocolo HyperText Transfer Protocol Secure (HTTP).Se realiza una descripción mas concisa del software necesario en el apartado4.5.

4.3. Compilación cruzada de código desde el gateway al nodo

El gateway actúa como nodo principal ejecutando un SO Linux. Pero los nodos sensores y actuadores se ejecutaran en pequeñas placas micro-controladoras en las que se cargara código que permitirá a las mismas conectarse a la red domótica y ejecutar sus tareas. Las placas `nodeMCU` seleccionadas en la creación del prototipo se basan en un firmware en el lenguaje de programación Lua. Sin embargo, para facilitar el desarrollo se crea el código en C, utilizando las múltiples librerías ya existentes en los entornos de placas de Arduino.

Para facilitar todavía mas el despliegue de estas placas en la APP que se va a desarrollar un script que con unos parámetros definidos, permite crear un código completo y listo para compilarse y subirse a una placa microcontroladora. Se puede plantear esta estrategia en base a que por distintos que sean los muchos modelos de sensores y actuadores compatibles con estas plataformas de hardware, la estructura del código es similar en todos los casos. Tanto si se compila un código para que una nodo informe por MQTT al gateway de una captura de datos de temperatura con un sensor, o si el servidor indica a un nodo con un actuador que entre en funcionamiento, el código que compone ambos casos cuenta con los mismos puntos comunes, estos son:

- Librerías de comunicación inalámbrica
- Configuración para la conexión inalámbrica y protocolos de comunicación con el servidor.
- Inicialización de la conexión con el servidor y procesos de recuperación de conexión.
- función de callback para responder a las peticiones del servidor.
- función de loop para tareas recurrentes o informes automáticos al servidor.

De esta forma, solo hay que preocuparse de las características individuales del hardware sensor o actuador que conectemos a la placa, el cual a su vez cumple con un patrón idéntico a todo los casos

- Librerías concretas para el hardware actuador o sensor.
- Inicialización hardware actuador o sensor.
- Procesos a ejecutar en el bucle y el callback del código.

Si se programan las partes del código común en ficheros separados, que incluyan en su interior marcadores de las partes particulares que corresponden al hardware del actuador

o sensor, y a su vez se programan estas particularidades también en secciones separadas, entonces un script debe ser capaz de juntar estas secuencias correctamente y concatenarlas en el orden adecuado dando un código valido mediante unos pocos parámetros.

El objetivo de este script es reutilizar las partes de código comunes a todas las posibles configuraciones posibles juntando las secciones particulares de una opción dada para unificar un único fichero coherente y valido para que el entorno de desarrollo de Arduino lo compile y suba a la placa dicho código de manera automática. Este fichero resultante del script se conoce en Arduino como sketch. El `sketch` es un fichero de extension `.ino` que contiene código en lenguaje C que puede ser compilado para ser subido en lenguaje Lua a una placa microcotroladora.

4.3.1. Proceso de generación de código para compilación cruzada de nodos

Crear un sketch mediante un script puede variar en su complejidad en el grado de cuantos argumentos reciba y qué tan flexible se desea que sea su comportamiento. Sea cual sea el tipo de sketch que se busque crear, se debe mantener una estructura de carga de librerías y constantes, un constructor y un bucle de ejecución. Considerando las necesidades del proyecto, se requiere que el script reciba los siguientes argumentos de entrada:

- Modelo de la placa microcontroladora en la que se sube el sketch.
- Modelo del sensor/actuador que va a conectarse en el GPIO de la placa microcontroladora.
- Valor numérico del pin de conexión de I/O en el que se conectará el sensor/actuador.
- Nombre de la estancia en la que se planea ubicar la placa microcontroladora.

Con esta estrategia, el administrador de la suite domótica puede añadir nuevos sensores y actuadores al abanico de opciones disponibles en la aplicación del usuario.

4.3.2. Generador de sketchs dinámicos para nodos

El objetivo ultimo del script es generar un sketch de Arduino que pueda compilarse y subirse a la placa correctamente; sin embargo, hasta que se reciben los argumentos sólo es posible disponer de una plantilla genérica con secciones que deben ser rellenadas en base a las opciones que se indicarán en argumentos más adelante.

El modelo de plantilla básico puede constituirse con código previamente escrito en diferentes ficheros para luego ser ordenadamente ensamblados en un único fichero de código. En el caso de este proyecto, se crea una estructura de carpetas constituida por un directorio principal, el script generador, un directorio con las secciones de código común a todos los sketch y directorios separados para cada sensor/actuador disponible en la aplicación.

Para la prueba inicial incluiremos el código del sensor DHT11 para un configurar una placa detectora de temperatura y humedad. En términos generales, un sensor/actuador requiere de 4 bloques de código que integrar en código común, las librerías y constantes, la configuración inicial del setup, la resolución de callback que atiende peticiones concretas del servidor y el código del bucle que será requerido por el protocolo MQTT para publicar de manera constante.

En la figura 4.2 se muestra como las diferentes secciones comunes a todos los códigos son combinadas con las secciones especificas de código en función de los argumentos definidos para el script, con el objetivo de crear un código valido para una placa con un sensor de temperatura y humedad, modelo dht11.

El proceso que debe preparase para que este esquema se cumpla requiere de la creación de directorios con los contenedores de códigos por partes, que serán necesarios para generar un sketch válido que pueda compilarse y subirse a un nodo, esta es la secuencia que genera

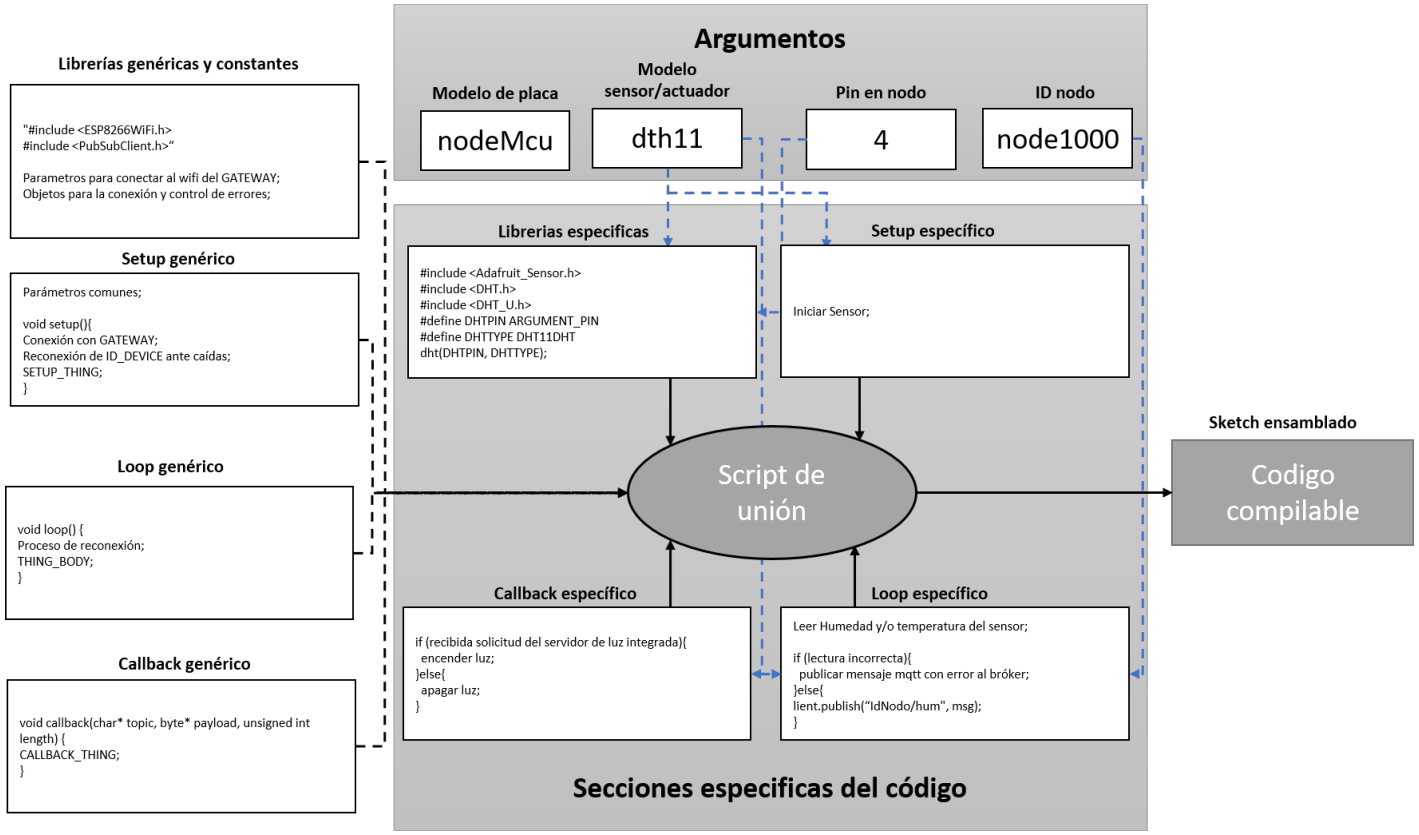

Figura 4.2: *Esquema de generación de sketch por partes*

dicha estructura:

```
cd ~
mkdir sketchgenerator
touch sketchgenerator/sketchgenerator
mkdir sketchgenerator/general
touch sketchgenerator/general/loop
touch sketchgenerator/general/setup
touch sketchgenerator/general/callback
mkdir sketchgenerator/dht
touch sketchgenerator/dht/dht11Lib
touch sketchgenerator/dht/dht11Loop
touch sketchgenerator/dht/dht11Setup
touch sketchgenerator/dht/dht11Callback
```

Con esta estructura, el script tendrá que construir un fichero de extensión .ino válido en la ruta de destino necesaria para que posteriormente arduino-cli pueda compilarlo y subirlo a la placa microcontroladora conectada a un puerto USB del nodo principal. Primero consideraremos el código de uso común a todos los sketch. Primero se creay y edita el fichero de configuración general. El segmento de codigo que se encarga de iniciar las comunicaciones con la red inalambrica, e inicializa el objeto sensor o actuador si es preciso. Es importante respetar la ruta de ficheros para que el generador de scripts funcione correctamente, en este caso, se ubica, partiendo del directorio contenedor del script, en `sketchgenerator/general/setup`

Considerando que el sensor DHT11 posee cierto código que se incluye en la función setup, se usará una cadena de caracteres a modo de marcador, el cual será sustituido por el código definido en un fichero concreto (dht11Setup, en este caso) situado en el directorio del sensor definido por los argumentos del script generador. Dicho centinela en este código

aparece como SETUP_THING. Adicionalmente, en este segmento, y demás casos, se incluirá el centinela ID_DEVICE que sera entregado en los argumentos del script para identificar una placa de manera inequívoca por parte del servidor.

```
//Variables para los mensajes de mqtt
long lastMsg = 0;
char msg[50];
int value = 0;

void setup() {
  setup_wifi();     //Inicia el cliente de conexiones
  client.setServer(mqtt_server, 1883);  //Vincula la comunicación mqtt con el servidor
  client.setCallback(callback);         //Establece una funcion en el observador de peti

  //Este es el marcado que sera sustituido por el codigo particular para este sensor.
  SETUP_THING
}

void setup_wifi() {
  delay(10);
  WiFi.begin(ssid, password); //Inicia la conexión inlamabrica con el servidor.
  while (WiFi.status() != WL_CONNECTED) {  //espera para intentos de conexión
    delay(500);
  }
}

void reconnect() {
```

```
  //Cuando la placa se conecta, se sucribe a un identificador
  while (!client.connected()) {
    //dicho identificador sustituira al marcador segun el argumento de script
    if (client.connect("ID_DEVICE")) {
      client.publish("ID_DEVICE/status", "hello world");
      client.subscribe("ID_DEVICE");
    } else {
      delay(5000);
    }
  }
}
```

Para que el servidor pueda realizar peticiones concretas a una placa, ésta debe resolver mediante la función callback los argumentos de entrada por MQTT cuando se invoque un topic al que esté suscrita dicha placa. En esta sección del código se incluye un centinela `CALLBACK_THING` que será sustituido por la lógica correspondiente a cada sensor/actuador. En general, esta sección de código está planificada para ser usado por actuadores que pueden tener distintos estados, como un relé (encendido o apagado). El siguiente fragmento de codigo se encuntra en la ruta `sketchgenerator/general/callback`.

```
void callback(char* topic, byte* payload, unsigned int length) {
  //En el callback solo es necesario poner el marcador que sera sustituido
  //Dicha sustitución corresponde al código concreto del sensor para este punto.
  CALLBACK_THING
}
```

El siguiente segmento de código común corresponde al bucle sin fin de un sketch. Al igual que en el caso anterior, una sección del código deberá ser rellenada por las particularidades del sensor. Para este caso se ha definido un centinela `MAIN_BODY`. El codigo del bucle se ubica en `sketchgenerator/general/loop`.

```
void loop() {
if (!client.connected()) { //recuperar caidas de conexíon
    reconnect();
  }
  client.loop();
  long now = millis();
  if (now - lastMsg > 2500) {
    lastMsg = now;
    MAIN_BODY   //Este marcador sera sustituido por código particular del sensor-actuado
  }
}
```

Para las secciones de código de un sensor como el DHT se requiere de las librerías, funciones auxiliares y constantes propias. La ubicación de los códigos específicos se crea para que responda a los argumento del script, en este caso `sketchgenerator/dht/dht11Lib`.

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
//Este amrcador sera sustituido por el valor del argumento correspondiente
#define DHTPIN ARGUMENT_PIN
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

También es necesario el segmento de código que se incluye en la función setup, ubicado en `sketchgenerator/dht/dht11Setup`.

```
dht.begin(); //para este caso, solo se inicia el objeto sensor.
```

La sección de callback de este sensor es limitada, ya que a la hora de atender peticiones expresas del servidor, como máximo se podría requerir las medidas del sensor, como

añadido funcional, podemos manipular la luz integrada del sensor. Siguiendo los anteriores segmentos, este se encuentra en `sketchgenerator/dht/dht11Callback`. No es necesario, pero ejemplifica como en un actuador funcionaria el callback, alterando el estado del mismo según corresponda. Este ejemplo particular seria valido para iluminar una fuente de luz.

```
if ((char)payload[0] == '1')
{
    digitalWrite(BUILTIN_LED, LOW);
}
else
{
    digitalWrite(BUILTIN_LED, HIGH);
}
```

Y por último el segmento de código que será invocado en la función loop que incluye la respuesta del cliente MQTT. Este segmento de código también incluye centinelas que deben ser sustituidos en base a los argumentos del generador de sketch, ya que de otra forma no sería posible definir los topics a los que responderá la placa cuando el nodo solicite información. Este segmento de codigo se ubica en `sketchgenerator/dht/dht11Loop`

```
float h = dht.readHumidity();   //el sensor registra en la variable el estado.
float t = dht.readTemperature();
if (isnan(h) || isnan(t))
{
    snprintf (msg, 75, "{'status':'error', 'message':'Error in DHT11 sensor'}", value);
    client.publish("nodemcudht11", msg);
}
else
{
```

```
    // Se publica en el canal deseado segun el identificador que se le asigne
    // segun los argumentos del script.
    snprintf (msg, 75, "'%f'", t);
    client.publish("ID_DEVICE/temp", msg);
    snprintf (msg, 75, "'%f'", h);
    client.publish("ID_DEVICE/hum", msg);
}
```

Con todo este código por segmentado puede constituirse un sketch válido siempre que se encadene en el orden correcto con las sustituciones adecuadas. El script se ejecutará en el entorno de `Bash`, y realizará la recepción de los argumentos, validación de los mismos y verificación de las rutas de los ficheros de donde se obtendrán los segmentos de código de cada parte. Serán unidos y escritos en un fichero situado en la ruta de un proyecto de Arduino que actuará como contenedor para las compilaciones. El script además dispone de los credenciales necesarios para la conexión WI-FI al nodo y el servicio de MQTT. A continuación se muestra el código del generador de sketch utilizado en el prototipo.

```
#!/bin/bash

# store arguments
args=("$@")
# get number of elements
ELEMENTS=${#args[@]}

# Determinar la ruta donde se creara el fichero para que el entorno de arduino-cli lo co
destinyPath=../Arduino/generatedino/generatedino.ino

BASICLIBRARIES="#include <ESP8266WiFi.h>\n#include <PubSubClient.h>"
```

```
echo -e $BASICLIBRARIES "\n\n"> $destinyPath

#$1 path to sensor
#$2 pint for I/O in sensor
[ $# -eq 0 ] && { echo "Usage: $1 sensor path"; exit 1; }
[ ! -f "$1" ] && { echo "Error: $1 file not found."; exit 2; }
file=${args[0]}
PIN="D${args[1]}"
DEVICE=${args[2]}
BOARD=${args[3]}

#Si fichero existe y no es vacio
if [ -s $file ]
then
        while IFS= read -r line
        do
        replace="${line/ARGUMENT_PIN/$PIN}"
        echo -e $replace >> $destinyPath
        done < "$file"
else
        echo -e "FICHERO SOLICITADO NO EXISTE\n"
        exit 2;
fi

ACCESS_POINT="const char* ssid = \"edomus\";\n"
PASSWORD="const char* password = \"sistemaguardian1970.\";\n"
MQTT_SERVER="const char* mqtt_server = \"192.168.4.1\";\n"
```

```
WIFI_CONST="WiFiClient espClient;\n"
WIFI_CLI="PubSubClient client(espClient);\n"



echo -e $ACCESS_POINT$PASSWORD$MQTT_SERVER$WIFI_CONST$WIFI_CLI >> $destinyPath

#GET SETUP FOR SELECTED THING
thingSetup=""
[ ! -f "$1Setup" ] && { echo "Error: $1Setup file not found."; exit 2; }
if [ -s "$1Setup" ]
then
        while IFS= read -r line
        do
        replace="${line/ID_DEVICE/$DEVICE}"
        thingSetup+=$replace
        done < "$1Setup"
fi
[ ! -f "general/setup" ] && { echo "Error: setup file not found."; exit 2; }
if [ -s "general/setup" ]
then
        while IFS= read -r line
        do
        main="${line/SETUP_THING/$thingSetup}"
        main="${main//ID_DEVICE/$DEVICE}"
        echo -e $main >> $destinyPath
        done < "general/setup"
fi
```

```
#GET CALLBACK FOR SELECTED THING
callbackSetup=""
[ ! -f "$1Callback" ] && { echo "Error: $1Setup file not found."; exit 2; }
if [ -s "$1Callback" ]
then
        while IFS= read -r line
        do
        replace="${line/ID_DEVICE/$DEVICE}"
        replace+="\n"
        callbackSetup+=$replace
        done < "$1Callback"
fi
[ ! -f "general/callback" ] && { echo "Error: setup file not found."; exit 2; }
if [ -s "general/callback" ]
then
        while IFS= read -r line
        do
        main="${line/CALLBACK_THING/$callbackSetup}"
        echo -e $main >> $destinyPath
        done < "general/callback"
fi

#GET LOOP BODY FOR SELECTED THING
thingLoop=""
[ ! -f "$1Loop" ] && { echo "Error: $1Loop file not found."; exit 2; }
if [ -s "$1Loop" ]
```

```
then
        while IFS= read -r line
        do
        replace="${line/ID_DEVICE/$DEVICE}"
        thingLoop+=$replace"\n"
        done < "$1Loop"
fi

[ ! -f "general/loop" ] && { echo "Error: loop file not found."; exit 2; }
if [ -s "general/loop" ]
then
        while IFS= read -r line
        do
        main="${line/MAIN_BODY/$thingLoop}"
        replace="${main/ID_DEVICE/$DEVICE}"
        echo -e $replace >> $destinyPath
        done < "general/loop"
fi
```

En este punto, tras ejecutar el script, se genera el sketch en la ruta definida para la combinación de argumentos dada. Sin embargo, los procesos de compilación y subida serán realizados por el software de `arduino-cli`. Es por ello, que debe incluirse en el script el siguiente bloque de comandos, que realizaran el proceso restante.

```
#compile the sketch
arduino-cli compile --fqbn $BOARD /home/kadaiser/Arduino/generatedino --debug

#fix permisions for device
```

```
sudo chmod a+wr /dev/ttyUSB0


#upload the sketch
arduino-cli upload -p /dev/ttyUSB0 --fqbn $BOARD /home/kadaiser/Arduino/generatedino
```

El comando del script para la configuración de prueba es:

```
./sketchgenerator dht/dht11 4 nodemcu823234 esp8266:esp8266:nodemcuv2
```

Si todo el proceso se ha ejecutado correctamente desde la generación del sketch, su compilación y subida a la placa, la correcta conexión de la placa a la red inalámbrica del gateway y su correcta publicación en MQTT, el servidor debería responder a la petición por parte del gateway con un JSON con la respuesta esperada.

4.4. Criterio de selección servicios

Al almacenar los datos generados por los nodos en el servidor, en los cuales una medida esta relacionada con un nodo que a su vez esta relacionado con una estancia, plantearse un modelo de Base de datos (BBDD) relacional es una buena opción. Pero si consideramos que, cada entrada almacenada tendrá una estructura distinta, hace que no sea la opción más ideal. Por ejemplo, utilizando una BBDD SQL se planifica un conjunto de tablas relacionadas entre sí. Es necesaria una tabla que contenga las ubicaciones y se relacione con otra tabla que defina a los nodos. Esto establece una relación 1:N donde múltiples nodos pueden existir para una estancia, pero nunca, un nodo podrá estar en varias instancias a la vez simultáneamente. De cada nodo existirá una nueva relación 1:N de medidas. Lo cual deja un esquema semejante al de la figura 4.3:

Existen algunos problemas graves de diseño de esta idea. En primer lugar, los dispositivos no pueden ser una propiedad de una estancia. Son objetos relacionados, pero no existe una `transitividad dura` entre ellos. Un nodo puede cambiar de estancia en un momento dado,

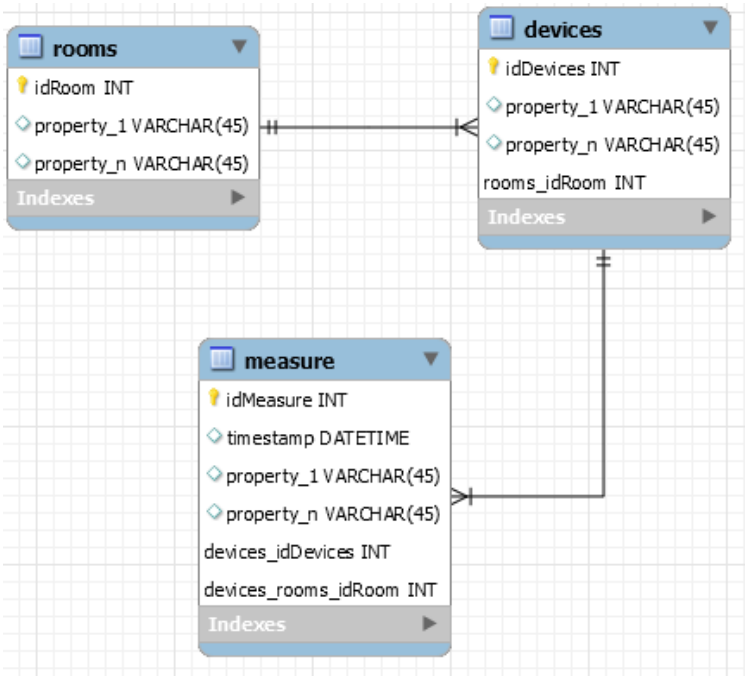

Figura 4.3: *Primer planteamiento de diseño de BBDD relacional*

y aun así, seguir existiendo medidas en fechas concretas de ese dispositivo para una habitación en la cual, dicho nodo ya no está relacionado. Otro posible escenario es la desaparición de una estancia (como resultado de fusionar 2 estancias en una al derribar una pared). Para mantener una integridad lógica y persistente a lo largo del tiempo. Toda medida deberá tener un campo que determine en que ubicación fue tomada. No hay transitividad dura entre room y device, lo cual hace que sean independientes entre sí. El siguiente esquema representado en la figura 4.4 de relación solventa este problema.

Con esto, aún hay que enfrentar un problema adicional. El número de columnas que definen las propiedades de una tabla. El ejemplo más claro, es la tabla de medidas. Una medida, efectuada por un nodo será definida por su identificador y la fecha en la que se realizó; ahora bien, según la naturaleza del dispositivo, se obtendrán distintos tiempos de medida. Un sensor combinado de temperatura y humedad nos dará dos magnitudes de medición, un sensor de ruido almacenará un valor de decibelios, una luz define su medida

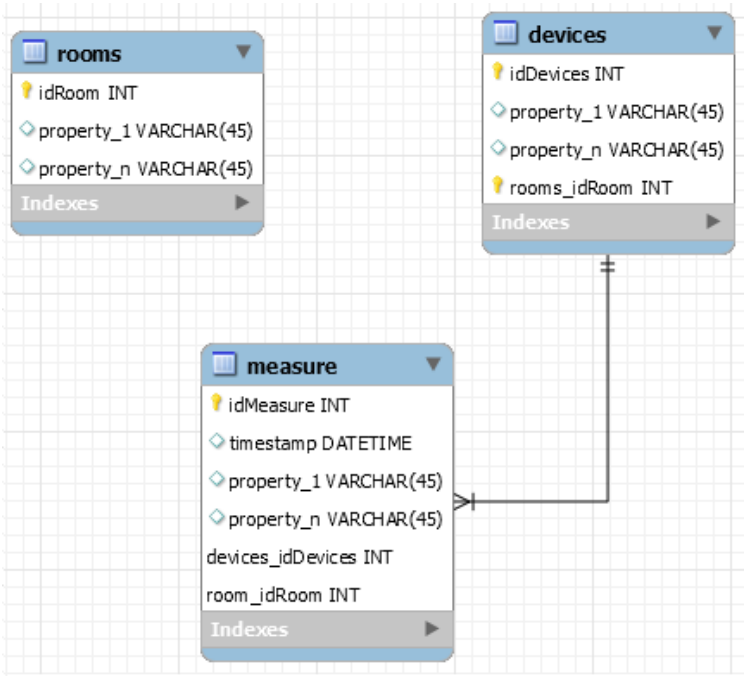

Figura 4.4: *Segundo planteamiento de diseño de BBDD relacional*

por su estado de actividad (encendido o apagado), aunque por otra parte podría indicar el consumo eléctrico, o propiedades adicionales como intensidad de luz, o incluso color (con conjunto de valores RGB). Es cierto que, para un actuador, como lo es un emisor de luz, no realiza medidas como tal, y sus correspondientes estados de actividad podrían ser más adecuados definirlos como propiedades del dispositivo y no como medidas. Podríamos separar las medidas de los estados en tablas distintas, pero igualmente llegaríamos al problema del número de campos necesarios en una tabla. Valor que por otra parte es muy difícil de prever en base a la extensa gama de dispositivos existentes. Esto puede solucionarse de manera sencilla con 2 estrategias.

1. Incluir una gran cantidad de columnas en previsión de los distritos tipos de medidas existentes, dejando que las medidas posean un valor nulo para los campos no utilizados en función de la relación de su sensor.

2. Integración de múltiples opciones de hardware compatibles con la suite domótica.

3. unificar todos los campos en un único valor de cadena de caracteres que almacene un dato estructurado, como es el caso de los JSON. Esta última opción, sería la más deseable tanto por sencillez de implementación como facilidad de procesamiento.

Lo que dejaría un esquema como el reflejado en la siguiente figura4.5.

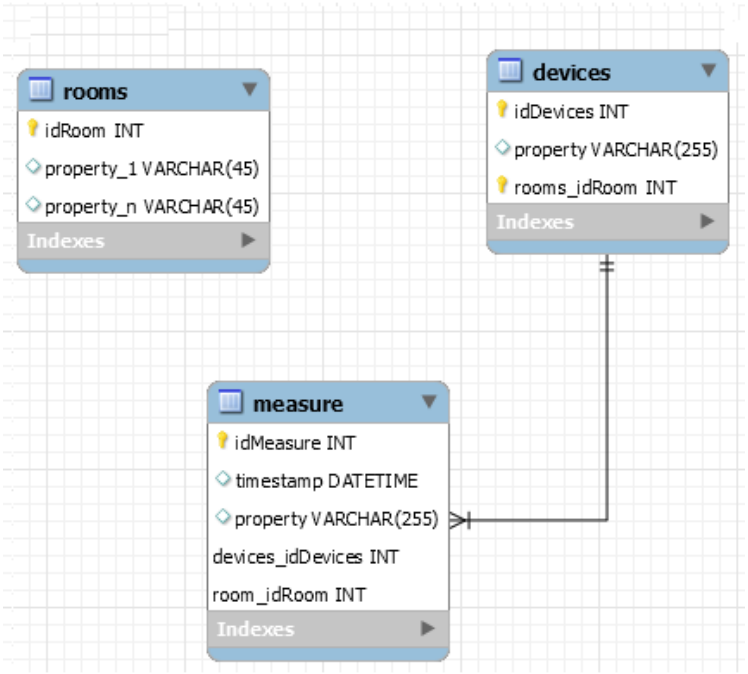

Figura 4.5: *Tercer planteamiento de diseño de BBDD relacional*

Una preocupación que agrava la perspectiva de usar una BBDD relacional es, en este punto del proyecto, su escalabilidad horizontal. Si bien las tablas pueden crecer a un gran número de registros, no se prevé almacenar datos con vistas a largo plazo, la mayoría de los valores almacenados serán efímeros en tiempo de utilidad, almacenarlos responde solo a la necesidad de obtener comparativas en plazos de tiempo relativamente cortos, como horas, días, y posiblemente semanas. Más allá de este rango estos datos no tienen una utilidad real y pueden ser condensados en medias para utilizarse en resúmenes. Por otro lado, disponer de flexibilidad a la hora de configurar la extensión de propiedades de un objeto de la BBDD es uno de los puntos fuertes de una BBDD no relacional.

4.5. Arquitectura de la aplicación frontend

4.5.1. Consideraciones previas en el desarrollo de una APP

El desarrollo de la app frontend se ha servido de dos frameworks para salvar ciertos escollos básicos no relacionados con la implementación de la suite domótica. Por un lado, se ha hecho uso del framework Open Source Ionic (en su versión 3), que es un framework Open Source para la construcción de aplicaciones híbridas multiplataforma, que nos permitirá portar la aplicación para cualquier dispositivo móvil Android a través de Cordova, y que además nos permitirá hacer uso de capacidades nativas de los dispositivos móviles (como el módulo de Wifi) gracias a los plugins existentes para ello. Además, Ionic proporciona grandes herramientas en cuanto a la maquetación responsive, lo cual ayuda en la creación de una aplicación móvil sin necesidad de invertir grandes esfuerzos en alcanzar un aspecto, usabilidad y elegancia mínimos y pudiendo redirigir ese esfuerzo a otros terrenos más fructíferos. Ionic está basado en Angular, el otro framework del que se ha hecho uso (en su versión 5) especialmente por su enfoque en una arquitectura basada en componentes que puedan ser reutilizados con mucha facilidad a lo largo de toda la aplicación. Las versiones modernas de Angular hacen uso de TypeScript para la programación de su lógica de negocio, lo que permite una mayor limpieza de código, mejores herramientas de tipado estricto y redunda en mayor escalabilidad. Angular se programa en TypeScript para la lógica de negocio, SASS para los estilos css y HTML para la maquetación.

4.5.2. Estructura básica

A grandes rasgos, la aplicación frontend está estructurada en:

1. Componentes, por lo general tendremos casi tantos como trozos o segmentos en los que se quiera desestructurar una vista (los conoceremos como *component*; serán útiles para aislar los diferentes comportamientos que se puede requerir en cada vista, y un mismo componente puede ser replicado infinitamente a lo largo de la aplicación cada vez con unos parámetros diferentes.

2. Módulos de vistas, generalmente uno por cada vista de la aplicación (lo que conoceremos como *view* o *template*); se encargarán de cargar la información que requiere la vista y de transformar las interacciones del usuario con la vista en flujos lógicos que generalmente pasan por (o acaban en) otros módulos.

3. Centros de gestión y manipulación de datos, generalmente uno por cada estructura de datos existente (lo que conoceremos como *store*); su cometido será proporcionar métodos a otros módulos o vistas para obtención y/o manipulación de su tipo de datos, siendo la store la responsable última de obtenerlos, manipularlos o almacenarlos por cualquier medio existente.

4. Servicios que ejercerán de cómodas APIs contra APIs externas o contra BBDD locales del dispositivo (los que conoceremos como *api provider/service* en el primer caso y *database service* en el segundo), generalmente uno de cada por cada estructura de datos existente; son responsables de lanzar las llamadas a dichos servicios con los parámetros adecuados (bien seas APIs REST remotas o BBDD locales) y controlar sus respuestas.

5. Servicios, que implementan utilidades y herramientas a lo largo de toda la aplicación de forma independiente, tanto que, técnicamente, podrían formar parte de cualquier otra aplicación (los que conoceremos como *service*); son responsables de aportar utilidades puntuales con la suficiente abstracción como para que el üsuario"de dicho servicio no conozca su funcionamiento en detalle, sino más bien, le resulte sencillo, conveniente y cómodo acudir a sus métodos.

Con el fin de comprender el esquema utilizado, mencionaremos los tipos de estructura existentes, aun cuando detallaremos su flujo más adelante: tendremos los dispositivos, que serán conocidos como *things* (siendo posiblemente lo más adecuado debido a su famosa acepción en el IoT); las habitaciones, que mencionaremos aquí y allá como *rooms*; los usuarios,

que conoceremos como *users*; las placas, que mencionaremos como *boards*; y los datos de aplicación generales, a los que nos referiremos como *application data*. Se observará que, en todos los casos, nos conviene referirlos en su traducción al inglés por facilidad a la hora de seguir el hilo del flujo en el código.

4.5.3. Patrón del flujo de datos

En pro de aplicar una arquitectura que intenta aplicar los principios de separación de responsabilidades y capas de abstracción, a continuación explicamos los patrones que sigue la estructura del proyecto frontend.

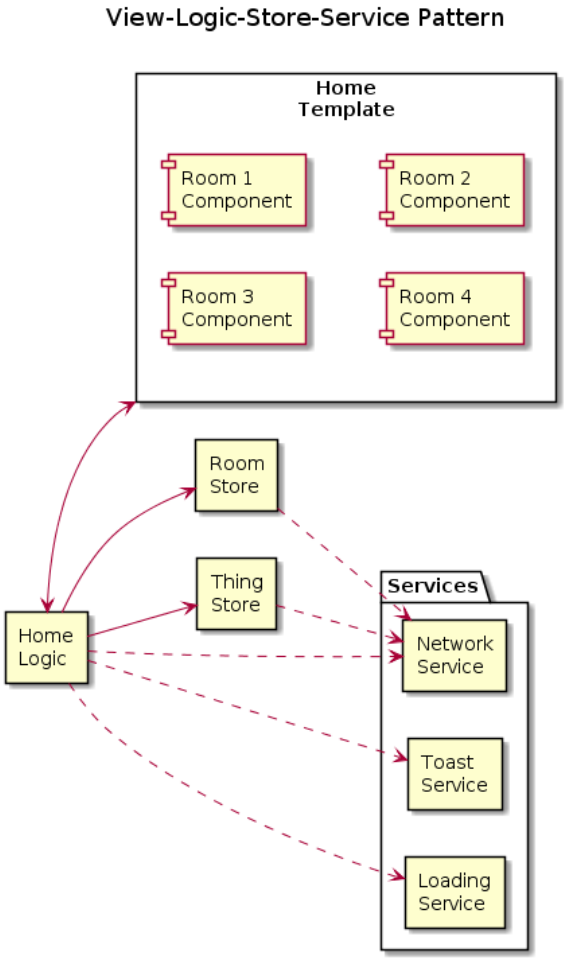

Figura 4.6: *View-Logic-Store-Service Pattern*

Por lo general, todas las vistas presentan, bajo un formato u otro, una serie de datos. El template será por tanto responsable de adaptar y mostrar estos datos como se requiera; pero primero deben obtenerlos, para lo cual delegarán en la store. A partir del momento en que se pide estos datos a la store, se crea una conveniente capa de abstracción que es útil en dos sentidos principales: por un lado, la vista no debe preocuparse del origen de dónde obtener esos datos, esto será responsabilidad de la store, lo cual hace más limpio el código del template, que debe preocuparse estrictamente de atender a los requisitos y cambios de la vista; por otro, la misma lógica que deba implementar la store para ofrecer estos datos a una vista, podrá ser reaprovechada para ofrecérselo a otra.

En el ejemplo de la Figura 4.6, podemos observar como el template de la vista *Home* debe mostrar una lista de todas las habitaciones existentes. Una vez el template ha adquirido este array de rooms a partir de la room store, deberemos mostrar un componente room por cada room en esa lista. Sin embargo, la longitud de esta lista no es conocida de antemano, y puede cambiar en cualquier momento. Por tanto, la implementación debe ajustarse a este dinamismo. Angular nos permite, mediante su directiva *ngFor*, generar tantos elementos como existan en un array de la lógica asociada al template, y además pueden ser elementos de cualquier tipo, luego esto nos sirve para crear una lista de componentes room, siendo posible instanciar cada uno con la información específica de cada elemento del array.

Observaremos que tanto la lógica asociada a una vista como las stores (así como otros services) hacen uso de los services. Estos services ofrecen, de forma autónoma e independiente, ciertas utilidades, como por ejemplo el LoadingService, que ofrece un spinner de carga con un mensaje de feedback para el usuario de que un proceso está cargando. El encargado de poseer y ejecutar la lógica que crea y muestra el spinner será el LoadingService; sin embargo, quien manda la orden inicial será siempre quien invoca al servicio, pues es quien conoce el estado actual de la vista: si los datos han sido pedidos y está a la espera de

recibirlos, pedirá mostrar el spinner; si finalmente se recibe confirmación de que la vista ya tiene disponibles los datos, se ordenará ocultar el spinner.

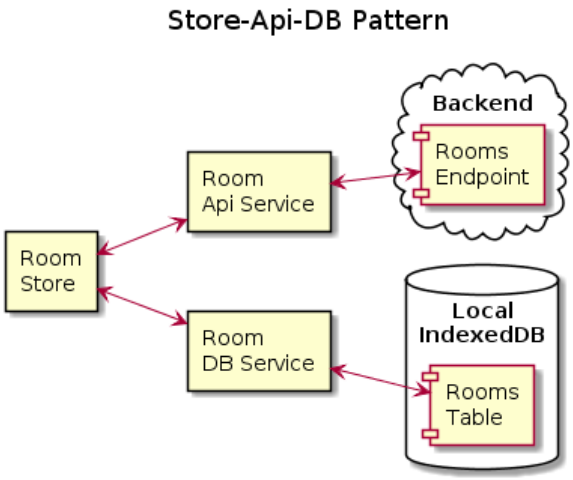

Figura 4.7: *Store-API-DB Pattern*

Respecto a la obtención de los datos, utilizamos el patrón asociado a la Figura 4.7. La store mantiene el rol más importante del flujo del frontend. Observaremos que, cada vez que un componente, vista o módulo desea interactuar con un ejemplar del tipo que regenta la store, por ejemplo, cada vez que un módulo desee obtener la información de una determinada habitación o desee cambiar uno de sus parámetros, pedirá esa información o ese cambio a la room store. Será en ese momento cuando la store decidirá de dónde obtener la información del objeto room, o a qué servicio externo (API o BBDD local) hacerle la petición de cambio. Así, veremos que la store hará las veces de distribuidor de la información de su tipo.

En este punto, con el fin de ejercer la separación de responsabilidades, hemos creado un módulo api provider (específicamente orientado para el mismo tipo de datos que la store) que permite a la store interactuar con la API remota de su mismo tipo; así, cuando la room store quiere obtener del backend la última lista real de rooms disponibles, hará uso del método `getAllRooms` del room provider; de la misma forma, cuando bajo petición del

usuario, quiera crear una nueva room, debe hacerse esta petición al servidor a través del método `createRoom` del room provider. Cada una de estas peticiones http a la API remota puede requerir de unas características particulares, como puedan ser su tipo CRUD, los headers http requeridos, la url particular de la petición y la estructura del cuerpo; y al delegar en el api provider, se evitar exponer a la store detalles intrínsecos a la petición http, que debería resultar comportarse como una caja negra de cara a la store, ya que no requiere ese nivel de detalle para su cometido de distribuir la información.

Es necesario apuntar que todos los api service que hemos creado (en general un por cada tipo de dato importante, o quizás más precisamente, uno por cada endpoint de entrada existente en la API remota), heredan de la clase ApiGenericProvider, la que implementa los métodos CRUD con los que interactuar con la API remota. En la Figura 4.8 se expone el diagrama de clases para todas las APIs creadas.

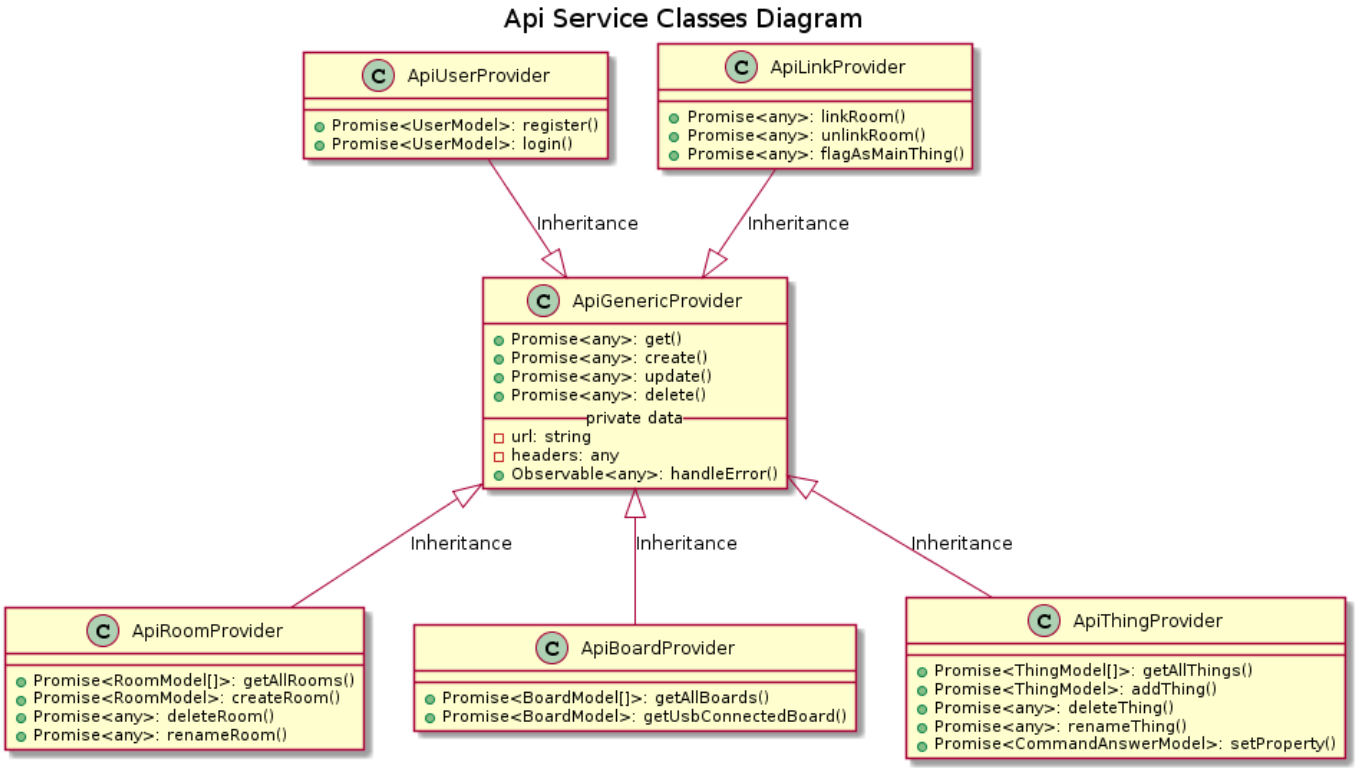

Figura 4.8: *API Providers Class Diagram*

De la misma forma y con el mismo objetivo que para el api provider, se crea un módulo database service (específicamente orientado para el mismo tipo de datos que la store), que permite a la store interactuar con la BBDD local para almacenar, recuperar o borrar cualquier entrada de ese tipo. En este proyecto, de entre las posibilidades existentes, hacemos uso de la base de datos IndexedDB, propia del motor web del dispositivo. Es una base de datos No-SQL que está orientada al almacenamiento de entradas de gran tamaño, y no bloquea la entrada-salida del hilo principal, de forma que es ideal para almacenar objetos JSON, que puedan tener estructuras y tamaños variables. En particular, cada database service se creará de forma que ataque específicamente a una única tabla, nombrada con el mismo tipo de datos, con el objetivo de reforzar la mantenibilidad del código y la separación de responsabilidades. Así, de nuevo, la store se permite desconocer si sus datos deben tener una estructura, codificación o formato en particular dependiendo de las características de la BBDD: delega esa responsabilidad en el database service, el cual debe garantizar que los datos sean devueltos a la store en el mismo formato en que fueron entregados en primera instancia.

Los módulos database service, a diferencia de los api providers, no heredan de un mismo módulo, pero queda pendiente como una mejora posible puesto que cada uno puede ser creado con una configuración particular (como ya ocurre) y, con la excepción de algunos métodos, se puede refactorizar todos aquellos que sean comunes en una clase padre. En la Figura 4.9 se expone la clase que sigue RoomDatabaseService, muy similar a la que siguen el resto de database services con sus configuraciones propias. Hemos adjuntado un diagrama de cada uno de ellos en el apartado 4.5.4.

4.5.4. Inicialización de la aplicación frontend

Cuando la aplicación Android se inicia, tras las inicializaciones propias del framework de Ionic y Angular, se ejecuta la lógica correspondiente a `app.module.ts`, el módulo base que se encarga de importar todo los módulos que se hemos implementado para nuestra aplicación

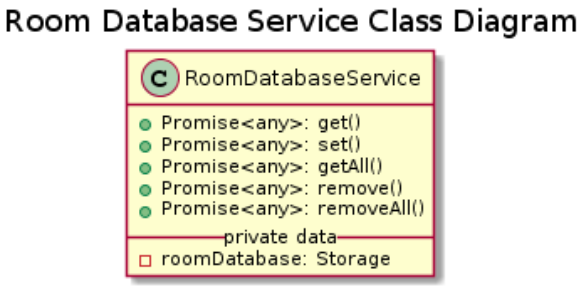

Figura 4.9: *Room Database Service Class Diagram-footnotemark*

antes de ejecutarlos por primera vez. El sistema de carga de módulos con *lazy loading* de Angular permite ejecutar algunos de ellos antes que el resto si, por necesidades de la aplicación, es condición necesaria que se procese antes que los demás. En nuestro caso, hacemos uso de esta herramienta para lanzar el método `initializeEndpoint` del módulo `core.module.ts`, que redirige al `initializeApiEndpoints` de `application-data.store.ts` y permite extraer de la BBDD local las IPs local y remota para poder disponer de ellas en la inicialización de los servicios de interacción con las APIs remotas.

Tras esto, se crea el `app.component.ts` que representa al componente base sobre el cual se lanza todo el sistema de páginas para cada vista y los módulos contenidos en todos ellos. En su constructor, se llama al método `initializeApplication` del `application.service.ts` (explicado más adelante) para que, en cuanto éste esté listo, oculte el *splash* inicial (en la aplicación Android) y podamos ver la página por defecto, la página de *Login*.

Describimos a continuación la inicialización de servicios y stores de la aplicación frontend, llevada a cabo en el `application.service.ts`. Los pasos siguientes ejecutan una serie de procesos asíncronos, pero la separación en pasos describe cómo cada proceso espera a que el anterior acabe, fin que logramos gracias a las promesas del `ECMASCRIPT 6`. Si alguno de los pasos finalizara erróneamente, el proceso de inicialización no continua y la aplicación no avanzará, debiendo así ser reiniciada.

1. Se ejecuta el método `_onCordovaEnvironment` que comprueba si la plataforma que

se está corriendo es una plataforma construida por Cordova, que nos permite ejecutar esta aplicación web en el dispositivo móvil Android (así como en otros), por lo que esta condición es necesaria para el correcto funcionamiento de toda la lógica del proyecto.

2. Se ejecuta el método `_initializeServices` que inicializa todos aquellos Services que utilizamos y deben ser inicializados. Primero lo hace con el `network.service.ts`, y cuando éste está listo, inicializa el `offline-reminder.service.ts` y acto seguido el `navigation.service.ts`. Se explica cada uno en detalle en su apartado en la siguiente sección.

3. A continuación, es necesario preparar todos los módulos que necesitan hacer cambios si el estado de autenticación del usuario cambia, con el fin de permitir (o no) a la aplicación mostrar al usuario rutas autorizadas o no (si el token de sesión expiró, se debe impedir que el usuario pueda acceder a información o comandos fuera de su permiso hasta que vuelva a conseguir un token válido). Así, se llama al método `listenAuthenticationStatus` de todas las stores y al `subscribeAuthenticationStatus` del `navigation.service.ts` el cual enrutará a la vista de Login si detecta que se ha perdido la autenticación del usuario, o enrutará a la vista de Home si detecta que ha conseguido un token válido.

4. Por último, se lanza el método `initializeUser` para iniciar sesión en la aplicación con un usuario previamente almacenado, si existe y tiene un token que no haya expirado todavía.

4.5.5. Diagramas de clases de las Stores y Models

La user store gestiona todo lo relacionado con el login del usuario, es por ello que tiene una fuerte cooperación con el AuthService. Durante la inicialización de la aplicación, se establece una suscripción a cambios de usuarios autenticados en el AuthService con el método público `listenAuthenticationStatus`.

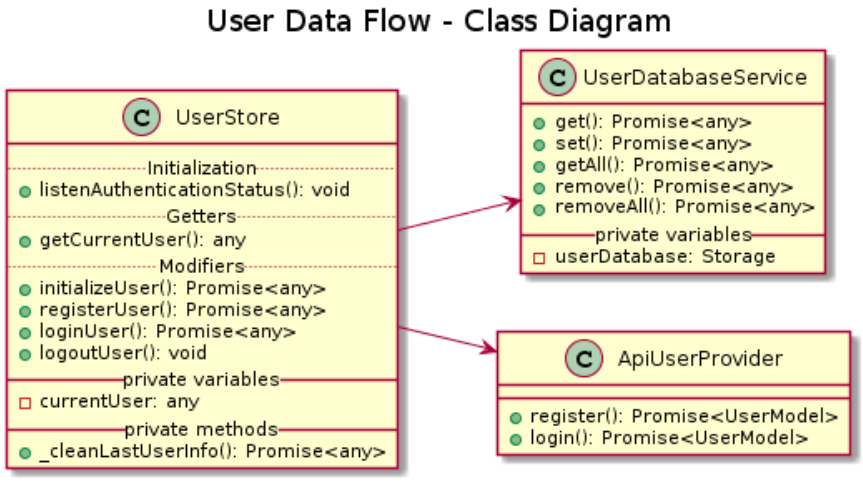

Figura 4.10: *User Data Class Diagram-footnotemark*

Nuestro proyecto implementa JWT de Auth0, para la autenticación de usuario mediante tokens, lo cual agiliza el login sin necesidad de que el usuario deba introducir de nuevo sus credenciales, y puede ser parametrizado desde el backend. Durante la inicialización de la aplicación, se lanza el método `initializeUser`, el cual busca en la userDB (local del dispositivo) la existencia de un JWT válido. Si lo encuentra, entonces puede pasarle el token al AuthService, el cual lo descifrará y extraerá el usuario autenticado, y establecerá el estado de autenticado del AuthService a verdadero. De esta manera, se la suscripción del `listenAuthenticationStatus` da lugar a pedirle a AuthService el usuario autenticado y se establece como el usuario actual. Si no, se lanza el método privado `_cleanLastUserInfo` para borrar cualquier información remanente en la aplicación móvil relativa al último usuario logado, acudiendo a todas las demás stores para que se encarguen de borrar de sus BBDD locales la información pertinente de la que son responsables.

Además de esto, la user store posee los métodos `loginUser` y `registerUser` que gestionan todo lo relacionado con los intentos de registrar y de iniciar sesión contra la API remota de user mediante el userProvider. Si logra logarse correctamente, el endpoint de login devuelve un token válido que la app almacena en la userDB y que utiliza para el propósito explicado anteriormente, además de pasar al AuthService este token válido para autenticar

al usuario.

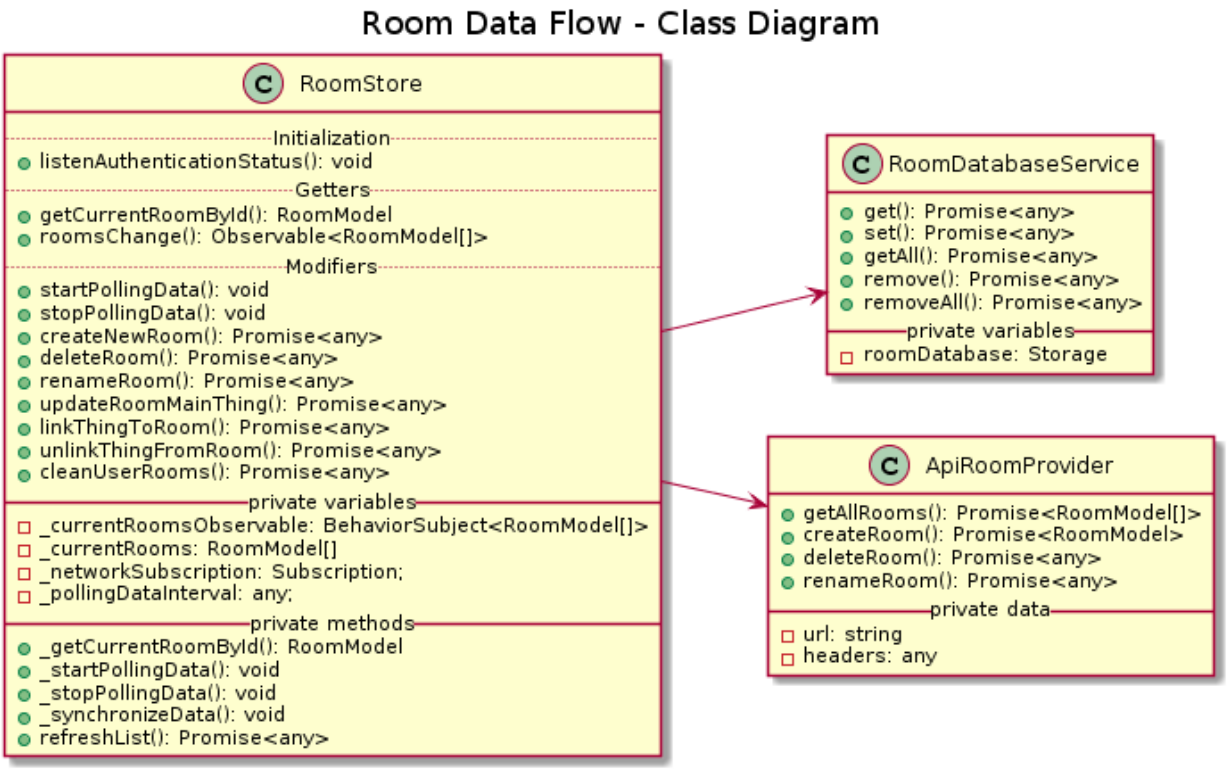

Figura 4.11: *Room Data Class Diagram-footnotemark*

La room store gestiona todo lo relacionado con la lista de rooms disponibles. Durante la inicialización de la aplicación, se establece una suscripción a cambios de usuarios autenticados en el AuthService con el método público `listenAuthenticationStatus`; y alinea esta suscripción con otra al estado de la red con el NetworkService, de forma que, si hay conexión a la red y además el usuario está autenticado, se llama al método privado `_startPollingData` (y de lo contrario, llama al `_stopPollingData`).

`_startPollingData` se encarga de llamar a intervalos de tiempo regulares de 5 segundos al método privado `_synchronizeData`, el cual hace uso del roomProvider para obtener la lista de todas las rooms. Cada vez que interactuemos con la API remota (con cualquiera de sus métodos, y aplicable al resto de stores), ejecutaremos un flujo que consiste en recibir la información, guardarla en la roomDB y llamar al método `refreshList`, que se encarga

de extraer el valor actual de esa lista de la roomDB y actualizar tanto el array privado `_currentRooms` como el observable `_currentRoomsObservable` con el nuevo valor.

Este último paso es crucial para el flujo de datos en la aplicación y representa uno de los patrones reconocidos más útiles de los que hacemos uso en la aplicación frontend: el patrón Observer/Subscribe. La librería RXJs nos ofrece este elemento tan importante, mediante el cual cualquier flujo de la aplicación que desee hacer uso de la lista de rooms, se suscribe al observable `_currentRoomsObservable` mediante el método `roomsChange` de la room store. Cada vez que la room store actualice este observable, el cambio será automáticamente notificado a todos los suscriptores de dicho observable.

La room store también ofrece los métodos `createNewRoom`, `deleteRoom`, `renameRoom`, `updateRoomMainThing`, `linkThingToRoom` y `unlinkThingFromRoom`. Los tres primeros son autodescriptivos, interactuan con la API y siguen el mismo flujo de actualización descrito anteriormente aunque adaptado a cada caso. Los tres últimos son llamados desde la thing store como parte de unos procesos encadenados, en los cuales se requiere que se cambie información específica de alguna room en particular, pero esta vez sólo de forma local (en el dispositivo), ya que la petición al backend realizada desde la thing store ya desencadena otros flujos secundarios en el backend que actualizan la información de la room implicada. Se garantiza la alineación de datos ya que el flujo está diseñado como si de una transacción se tratara: el backend siempre tiene la información verídica, y si ocurriera algún fallo en el flujo (sea en backend o en frontend), se rechaza el flujo para evitar datos incorrectos, que en cualquier caso serán actualizados con la siguiente petición `_synchronizeData`. La room store también expone el método `cleanUserRooms`, accedido y usado desde la user store para el borrado de datos de rooms.

La thing store gestiona todo lo relacionado con la lista de things disponibles. Tanto la suscripción al AuthService, y al NetworkService, como el sistema de petición periódica y recuperación de datos, así como la exposición del observable a posibles suscriptores, es idéntico al de room store pero evidentemente aplicado a la lista de things, por lo cual no

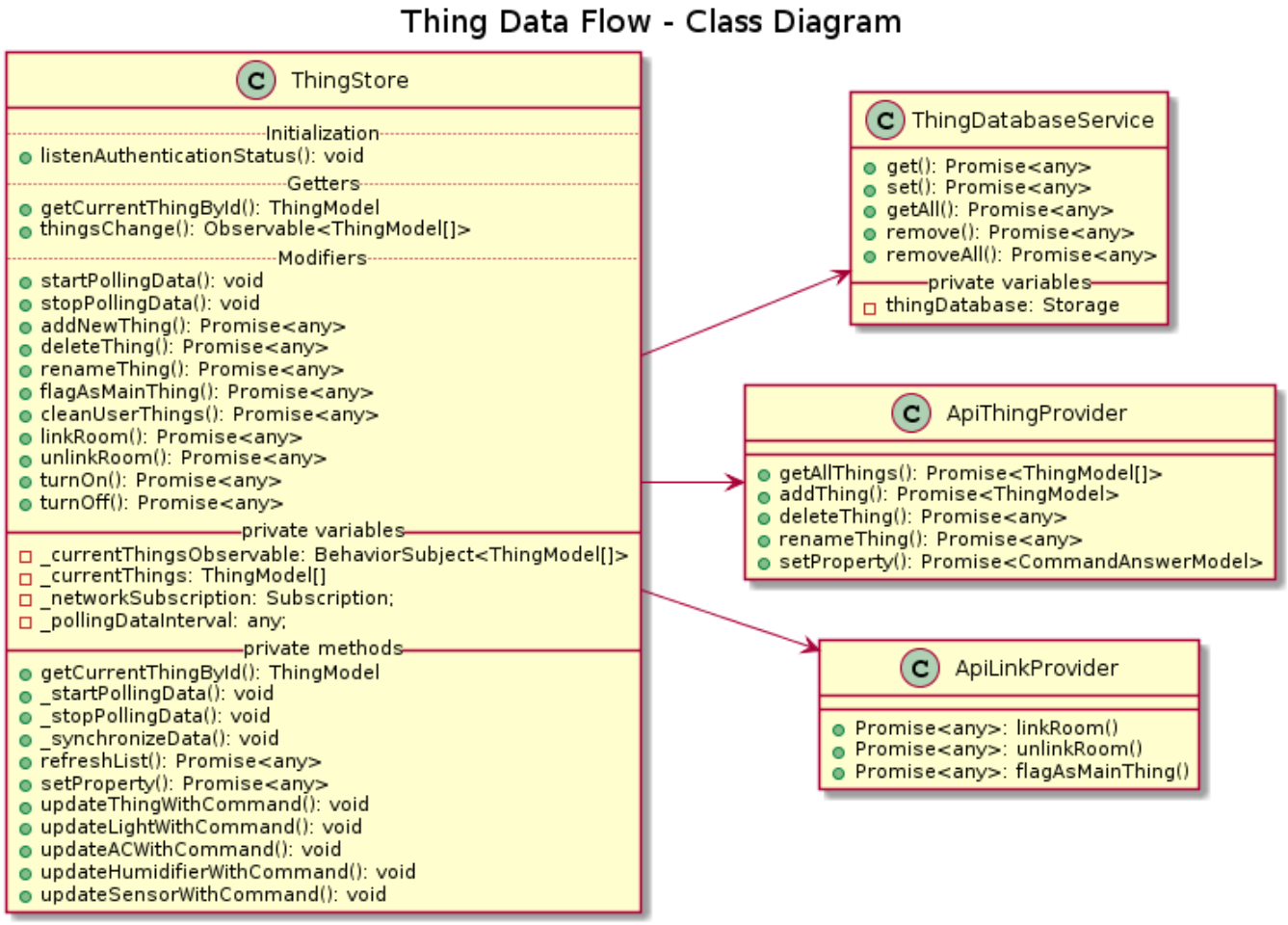

Figura 4.12: *Thing Data Class Diagram-footnotemark*

entraremos en detalle de nuevo.

La thing store también ofrece los métodos `addNewThing`, `deleteThing`, `renameThing`, `flagAsMainThing`, `linkRoom` y `unlinkRoom`, que interactuan con la API en sus métodos homónimos y siguen el mismo flujo de actualización descrito anteriormente aunque adaptado a cada caso. Los tres últimos

La thing store también expone el método `cleanUserThings`, accedido y usado desde la user store para el borrado de datos de things.

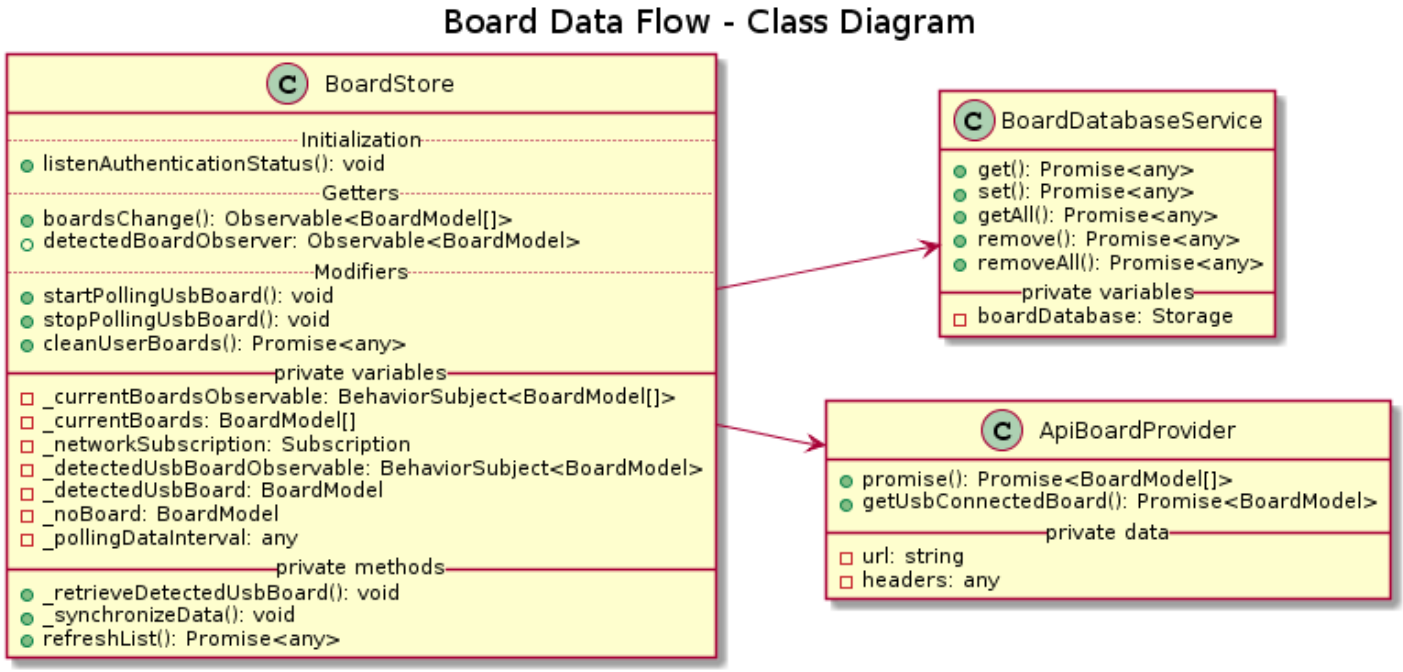

Figura 4.13: *Board Data Class Diagram-footnotemark*

La board store gestiona todo lo relacionado con las boards disponibles. La implementación actual, para el objetivo del proyecto, no requiere almacenar una lista de boards disponibles como ocurre con las stores de rooms o things; sin embargo, dispone de todo el flujo y todos sus métodos, al igual que con las otras stores, lo cual puede resultar útil para futuras ampliaciones del proyecto. El aspecto más importante de esta store es su flujo de detección de una placa conectada al puerto USB de la Raspberry Pi, preguntando periódica-

mente al backend, con el fin de comunicárselo a los módulos interesados en ejecutar alguna acción dependiendo de esto. Este flujo es explicado en contexto en el caso de uso 5.2.4.

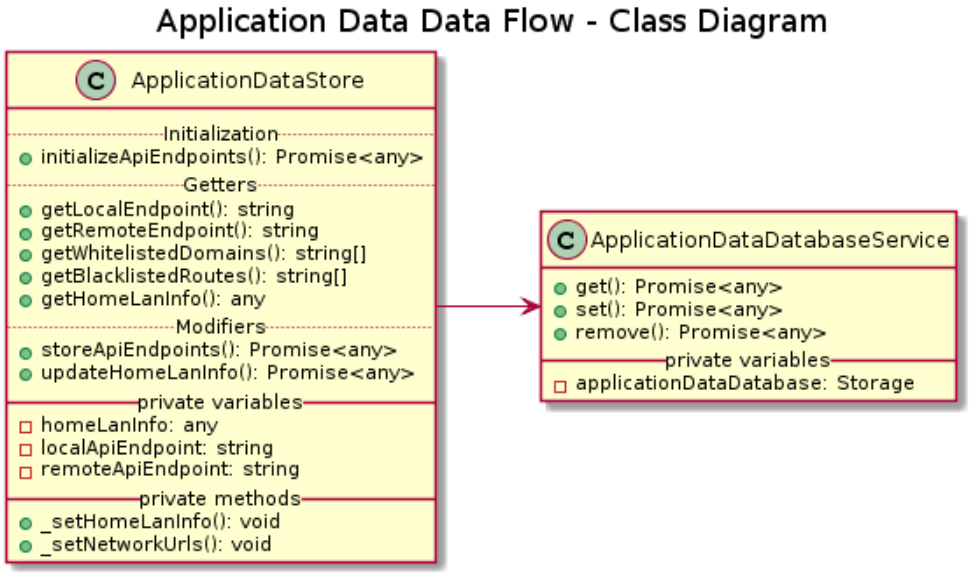

Figura 4.14: *Application Data Class Diagram*

La application data store gestiona toda la información general que la aplicación requiere almacenar y utilizar y que puede varia con las acciones del usuario; principalmente se encargará de almacenar y modificar las URLs locales y remotas a las cuales la aplicación debe apuntar, según las circunstancias.

Si el usuario se encuentra en la misma red local que la Raspberry Pi, mediante una acción del menú *Settings* podrá detectar la red actual y recordarla (IP local y SSID), mediante el método de la store `updateHomeLanInfo`, para así comunicarse por Wifi con el nodo principal, pudiendo prescindir de una conexión de Internet y la respectiva cobertura. Si por el contrario, se encuentra usando una red de datos (o una red Wifi desconocida, en la que no se encuentra la Raspberry Pi), utilizará la URL que apunta a la IP externa en la cual encontrar a la Raspberry Pi. Para setear las URLs a usar por los servicios de API remota, la store lanza el método `_setNetworkUrls` que cambia la info utilizada en el `network.service.ts`, el punto que utilizan dichas APIs. sus métodos `getLocalEndpoint` y `getRemoteEndpoint`. De

la misma forma, es la store encargada de ofrecerle al módulo JWT las rutas permitidas y las prohibida, mediante los métodos `getWhitelistedDomains` y `getBlacklistedRoutes` respectivamente.

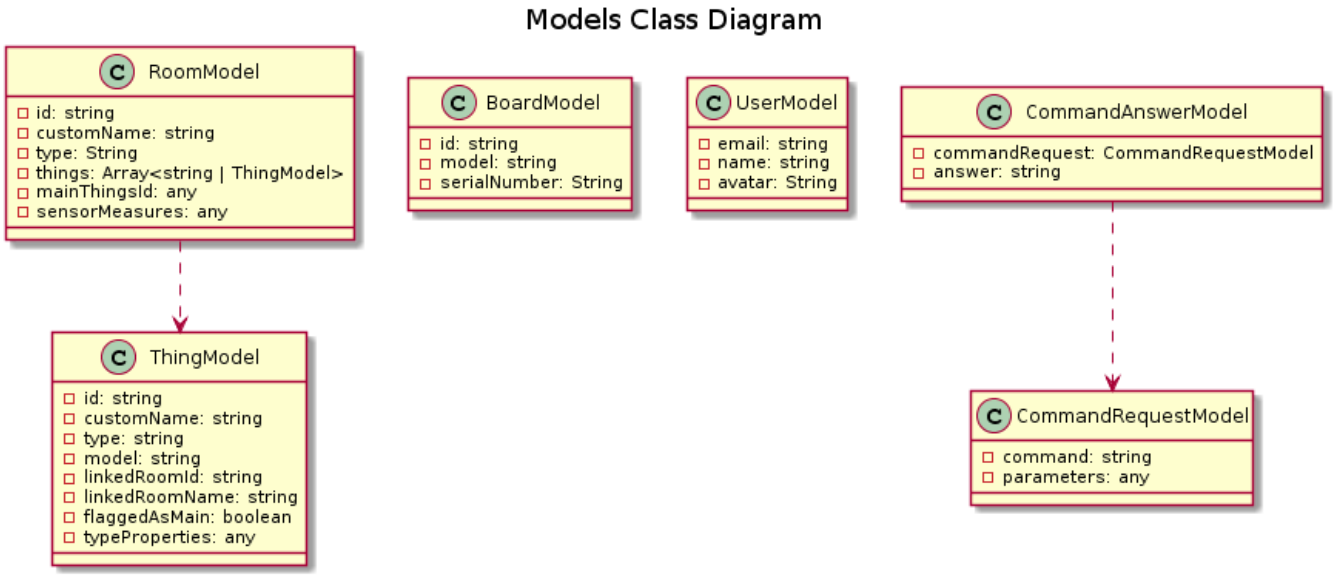

Figura 4.15: *Models Classes Diagram*

Los diagramas de clase responden a modelos para cada tipo de datos que sean claros y concisos, permitiendo toda la operatividad requerida para los casos de uso existentes.

1. Para la clase `RoomModel`, se requiere un `id` como identificador único de la room, un `customName` como ïdentificador humano"para el usuario, el `type` para categorizar el tipo de estancia (útil a la hora de presentar las habitaciones y discernir mediante una imagen y un símbolo el tipo de habitación a simple vista), un campo `things` para almacenar la lista de things asociadas a dicha room (es de tipo variable ya que en ciertos momentos de su flujo, serán simplemente identificadores, y en otros, serán las `ThingModel` como objetos completos), un objeto `mainThingsId` donde señalar la thing principal de cada tipo para dicha room y un campo `sensorMeasures` donde almacenar los cálculos generales extraidos de sus sensores asociados a la room.

2. Para la clase `ThingModel`, se requiere un `id` como identificador único de la thing, un `customName` como ïdentificador humano"para el usuario, el `type` para categorizar el tipo de thing (si es de tipo luz, de tipo sensor, etc.), un campo `model` para distinguir el modelo específico que rige el comportamiento de dicha thing, un campo `linkedRoomId` y otro `linkedRoomName` para marcar la room a la que está asociada la thing, un booleano `flaggedAsMain` para mostrarlo y distinguirlo del resto de things del mismo tipo asociadas a la misma room, y un `typeProperties` donde van todas las características, valores y detalles de dicha thing.

3. Para la clase `UserModel`, sólo es requerido su `email` como campo único, siendo su `nombre` y `avatar` opcionales para mostrar el perfil del usuario logado.

4. Para la clase `CommandRequestModel`, usamos un campo `command` que marca el comando a ejecutar sobre una determinada thing y el campo `parameters` marca los parámetros opcionales que requiere dicho comando.

5. Para la clase `CommandAnswerModel`, el campo `commandRequest` es del tipo `CommandRequestModel` y contiene el objeto comando para el cual este objeto es la respuesta al comando, así como un campo `answer` opcional que pueda añadir algún detalle o información de respuesta.

4.5.6. Diagramas de clases de los Services

Tal y como se describió en la sección 4.5.4, el `application.service.ts` tiene un rol importante de apoyar en la inicialización de la aplicación, encargándose él de inicializar tanto stores como services que lo requieran. No entraremos en mayor profundidad para explicarlo ya que el contexto de dicha sección contribuye a un mejor entendimiento.

El `auth.service.ts` se encarga de guardar y modificar el estado de autenticación de usuario en la aplicación, apoyándose en la librería `angular-jwt`. Expone métodos como

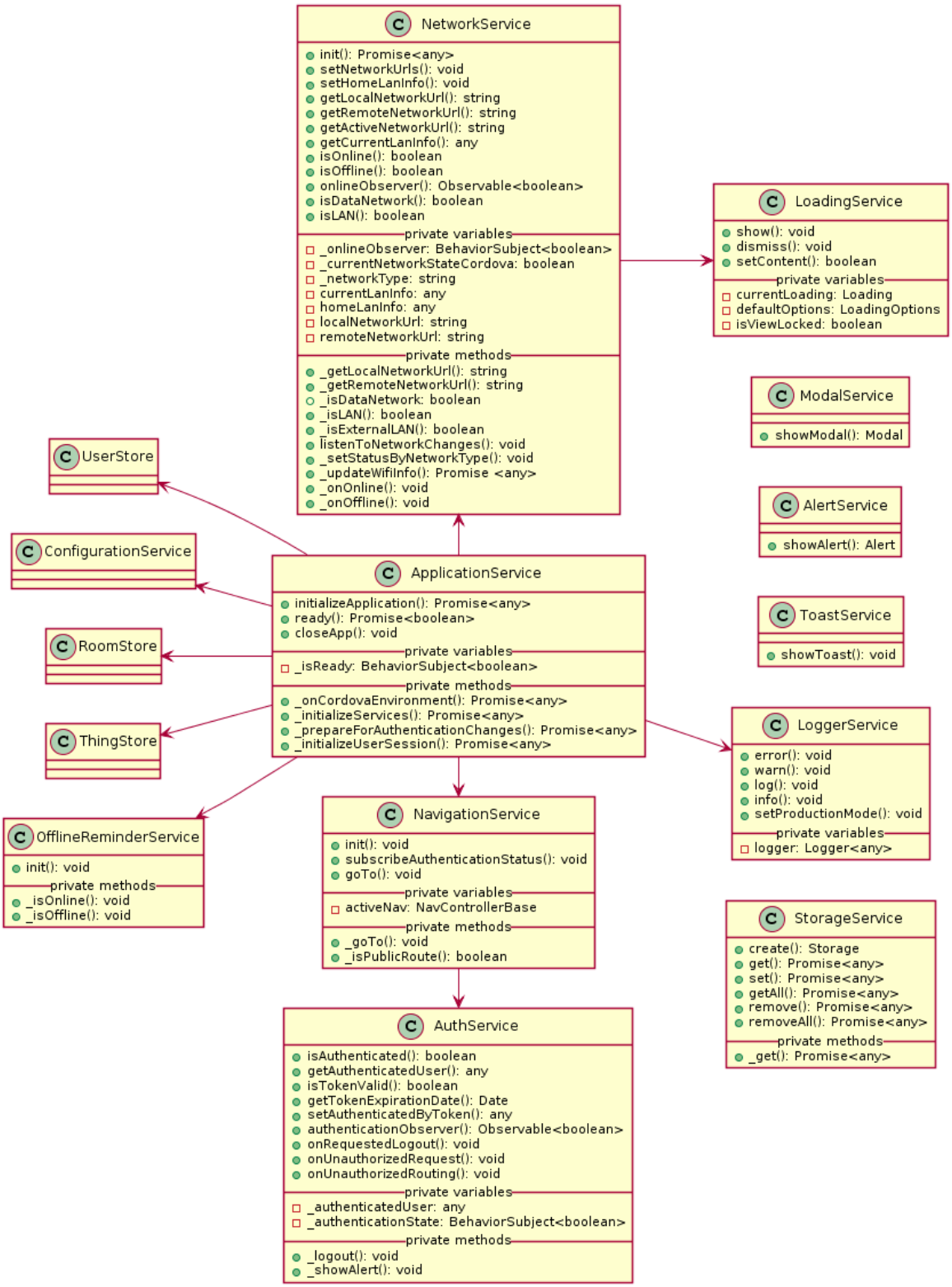

Figura 4.16: *Services Class Diagram*

`isAuthenticated`, `getAuthenticatedUser`, `isTokenValid` y `getTokenExpirationDate` para proporcionar información del estado actual de autenticación a otros módulos, así como el Observable `authenticationObserver` al cual se suscriben todas las stores en la inicialización de la aplicación, y el `navigation.service.ts` explicado más tarde, para recibir inmediatamente cualquier cambio de estado. También expone el método `setAuthenticatedByToken` para establecer un token como identificación de autenticación del usuario y cambiar el estado de autenticación a verdadero. Finalmente, ofrece un método por cada caso en el que se desea pasar al estado de autenticación falso, como puede ser `onRequestedLogout` para una situación de cierre de sesión, `onUnauthorizedRequest` para una petición ïlegal"(es decir, una petición que recibe una respuesta de token inválido), y `onUnauthorizedRouting` para un cambio de vista ïlegal"(es decir, enrutar a una vista privada cuando el estado de autenticación no debería permitirlo).

El `navigation.service.ts` es un módulo que cubre las funcionalidades para enrutar a diferentes páginas o vistas de la aplicación, utilizando el controlador de navegación de Ionic `NavControllerBase`. En la inicialización de la aplicación, se suscribe a los cambios de autenticación de autenticación, con el `authenticationObserver` del `auth.service.ts`: si se recibe una autenticación válida, se enruta automáticamente a la vista Home; si se recibe una inválida, a la vista de Login.

Su método estrella expuesto es el `goTo`, al cual se le pasa una ruta de página y parámetros de navegación y acudiendo al método interno homónimo `_goTo` recorre el siguiente flujo: si la ruta es pública (es decir, no se requiere estar logado para visitar la ruta), introducirá una nueva entrada en el historial de navegación (o si se trata de la ruta de Login, reiniciará el historial, por tratarse de la casuística de un cierre de sesión de usuario); si la ruta no es pública, comprobará si el usuario está autenticado, en cuyo caso introducirá una nueva entrada en el historial de navegación, y en caso contrario, avisará al `auth.service.ts` de

un enrutado sin permiso y se dirigirá a la raíz (página de Login).

El `network.service.ts` se encarga de controlar el estado de la red para proporcionar utilidades relacionadas a diferentes módulos de la aplicación, desde informar sencillamente de si estamos conectados a una red LAN o una conexión a datos 4G o sencillamente sin conexión alguna, como proporcionar la URL adecuada a la que lanzar las peticiones del backend cada vez que pueda cambiar la red a la que estamos conectados.

En su constructor, se encarga de comprobar si estamos en un dispositivo móvil al cual tenga que solicitar permisos para obtener información de la red WIFI a la que está conectado, como es el caso de Android, así que a través del módulo `AndroidPermissions` de Ionic, se podrá solicitar al usuario el permiso a conocer el estado de su red de WIFI y a su GPS. Contrariamente a lo que pueda parecer, es sólo un empaquetamiento de los permisos en Android lo que requiere conocer la información de GPS. Este módulo de la aplicación sólo utiliza el permiso para acceder a los datos de SSID y MAC de la red, y para ello utiliza la librería externa `WifiWizard2` con sus métodos `getConnectedSSID` y `getConnectedBSSID` respectivamente.

Al inicializarse lanza su método privado `listenToNetworkChanges` para detectar los cambios de red gracias al plugin de `Network` de Ionic, y reaccionar a los cambios ejecutando su método privado `_setStatusByNetworkType` y los métodos `_onOnline` y `_onOffline` dependiendo de si logra conexión a una red o no. Una vez conecta a una red, si es una red LAN, capturará la información de la red WIFI con su método privado `_updateWifiInfo` para almacenarla para futuras comparaciones con red preferida guardada en la application data store, y sólo entonces informar mediante el patrón Observer/subscribe a todos los módulos interesados en el cambio de red.

El módulo expone los siguientes métodos:

1. El método `setNetworkUrls` para establecer las URLs local y remota.

2. El método `setHomeLanInfo` para establecer la información de la red en la que está el nodo principal.
3. El método `getLocalNetworkUrl` para devolver la URL local.
4. El método `getRemoteNetworkUrl` para devolver la URL remota.
5. El método `getActiveNetworkUrl` para devolver la URL de la red activa actualmente.
6. El método `getCurrentLanInfo` para devolver la información de la red LAN actual.
7. El método `isOnline` para devolver si el estado actual es conectado a una red.
8. El método `isOffline` para devolver si el estado actual es desconectado de cualquier red.
9. El método `onlineObserver` para devolver un objeto observable al cual suscribirse que notifica los cambios de estado de la red.
10. El método `isDataNetwork` para devolver si la red actual es una red de datos.
11. El método `isLAN` para devolver si la red actual es una red LAN.

El `toast.service.ts`, el `modal.service.ts` y el `alert.service.ts` son servicios que aprovechan las funcionalidades de Ionic de *toasts*, *popups* y *alertas* respectivamente, aportando en sus constructores las configuraciones base necesarias para generarlos según nuestra preferencia; y exponen respectivamente sencillos métodos para su uso cómodo a lo largo de la aplicación, `showToast`, el `showModal` y el `showAlert`.

Otro servicio de la misma categoría sería el `loading.service.ts`, que sin entrar en detalles de implementación, ofrece el típico componente mostrado en pantalla de carga con o sin mensaje de espera, y que está implementado para ser el mismo por toda la aplicación; así,

cuando un módulo decide mostrar un mensaje de espera (con el método expuesto `show`), pero otro decide que existe una prioridad mayor para cerrarlo (con `dismiss`) o sustituirlo (con `setContent`), puede interactuar con el mismo sin problemas de apilación de componentes.

En la misma línea, el `logger.service.ts` proporciona un servicio de log para el proceso de *debug* de la aplicación durante su desarrollo, que permite personalizar altamente las características del log según el tipo de mensaje que se quiera apuntar, por ser un mensaje de error (logable usando el método `error`), un aviso (con el método `warn`), un sencillo log (con el `log`) o una información (con `info`).

El `storage.service.ts` es el utilizado por todos los database services de la aplicación para crear un almacenamiento particular para cada uno de ellos con `create`, y poder gestionarlo con los métodos `get`, `set`, `getAll`, `remove` y `removeAll`. Como módulo interno, hace uso del servicio de `Storage` de Ionic, que por detrás aprovecha todas las BBDD disponibles en el motor del navegador en uso, dando preferencia y usando en la mayoría de casos, la `IndexedDB`. Además, cabe destacar que, gracias a la librería `crypto-js`, hemos añadido otro nivel de seguridad configurable que se encarga de encriptar o desencriptar cada entrada al almacenarla o recuperarla respectivamente.

4.6. Arquitectura del servidor de NodeJS

4.6.1. Consideraciones previas

El desarrollo de la aplicación backend ha seguido la línea del stack MEAN, esto es, MongoDB, ExpressJS, Angular (éste sólo en el frontend) y NodeJS. Los beneficios de seguir dicho stack son sencillamente los de sus frameworks y herramientas: MongoDB aporta los beneficios de una BBDD no relacional, enormemente alineado con las características del proyecto, según el cual podríamos almacenar grandes cantidades de datos con diferentes

estructuras JSON bajo la misma lista de documentos sin perder velocidad de acceso o escritura; ExpressJS nos permite elaborar una API REST completa salvando los principales escollos que NodeJS podría presentarnos para tal fin, pues sus herramientas base para el tratamiento de peticiones http se presentan intrincadas y poco intuitivas; y NodeJS se beneficia de infinitas librerías de sencillo uso, constantemente probadas y actualizadas por su amplia comunidad, proveyéndonos en este caso de muchas utilidades que nos ayudan a evitar desarrollos difíciles e innecesarios no intrínsecos a una suite domótica (como es nuestro fin) y cuyos beneficios enumeraremos más adelante.

4.6.2. Estructura básica

A grandes rasgos, la aplicación backend está estructurada en:

1. Enrutadores, por lo general tendremos uno por cada nombre en nuestra API REST (los conoceremos como *routers*); gracias a ExpressJS, permiten encaminar cada petición http recibida en el servidor según su tipo CRUD (GET, POST, PUT o DELETE) y la estructura de la URL asociada, para ser atendida por el método específico que debe procesar dicha petición.

2. Controladores, por lo general tendremos uno por cada nombre en nuestra API REST (los conoceremos como *controllers*); encuadran un contexto en el que se crea, manipula, sirve, guarda y destruye elementos de la estructura de datos asociada, además de exponer una serie de métodos, generalmente uno por cada posible entrada de la API para el nombre de dicha API asociado a la estructura de datos en uso.

3. Modelos, de los que tendremos uno por cada estructura de datos existente (los conoceremos como *models*); definirán la estructura JSON que debe seguir un objeto para ser incluido en una collección de MongoDB, y que utilizaremos para generar y reconocer objetos de dicho tipo de estructura de datos y poder mejorar la interacción con la BBDD de MongoDB.

4. Servicios, que al igual que en el proyecto frotend, ejercerán de cómodas APIs contra diferentes servicios, como pueda ser la BBDD de MongoDB, el protocolo MQTT y otros módulos de utilidades (los conoceremos como *providers* o *services*); por lo general, podrían ser prácticamente independientes y ser copiados tal cual a cualquier otro proyecto (a falta de ligeras adaptaciones que en lineas generales facilitan su explotación en cada caso) y proporcionan una capa de abstracción adicional que delimita la responsabilidad del módulo que hace uso de dicho servicio y reduce su complejidad de código.

5. Módulos de soporte, por lo general tendremos tantos como sean necesarios, y cada uno se dedica exclusivamente a un tipo de estructura de datos (los conoceremos como helpers); ofrecen sus métodos tanto a su controller asociado como a otros controllers, de forma desacoplada con el fin de aligerar la carga de código del controller asociado.

6. Módulos intermediarios, que aportan utilidades adicionales a la hora de procesar peticiones http (los conoceremos como *middlewares*); gracias a ExpressJS, permitirán enriquecer el tratamiento de dichas peticiones para establecer filtros adicionales o transformaciones de los datos de la petición.

7. Archivos de configuración y de constantes, que permitirán aislar adecuadamente el uso de constantes útiles a lo largo de toda la aplicación, de una forma centralizada, independiente y de fácil acceso y modificación.

Con el fin de comprender el esquema utilizado y establecer una alineación con el frontend, la nomenclatura utilizada para cada tipo de datos será la misma que en el proyecto cliente.

4.6.3. Flujo de enrutado de las peticiones HTTP

Como hemos explicado anteriormente, ExpressJS es utilizado para el enrutado de toda petición HTTP que alcance la aplicación backend. Durante la inicialización del servidor, mediante el método `listen` en el `server.js` se utiliza el objeto `app` expuesto en el `src/app.js`

(un objeto generado por el framework ExpressJS) para generar un servidor que atenderá todas las peticiones HTTP en el puerto que se le configure, en este caso, se ha designado el puerto 3000.

Posteriormente, se lanza el setup del enrutador con el método `setupRouting` del módulo `router.js`, el cual se encarga de importar todos los routers existentes y vincularlos a cada uno de los endpoints que serán atendidos mediante el método `use` del objeto `app`. Así, suponiendo que se recibe una petición http con la ruta `/api/thing`, ésta será redirigida al router `thingRouter` mediante el método `use` y con los parámetros `ROUTER_CONFIG.EP_GLOBAL.THINGS` y `thingRouter`. El primer nivel de encaminamiento de las peticiones es tal que el mostrado en la Figura 4.17.

Una vez la petición ha sido encaminada a un router en particular, entramos en el segundo nivel de encaminamiento. En este punto, el router se encarga de importar el controller específico para su tipo de datos y de encaminar cada endpoint a la función final de un controller que atenderá a dicha petición. ExpressJS nos permite registrar dicha función a ejecutar para unas características particulares de petición, de forma que con unos sencillos métodos, podremos asociar dicho método dependiendo del tipo CRUD de la petición atendida y de las características de su URL. Así, suponiendo que nuestro `thing.router.js` atiende una petición PUT en la url `/api/thing/rename/:id`, mediante el método `put` podremos encaminar dicha petición a ser atendida por el método `renameThing` del `thing.controller.js`. Este segundo nivel de encaminamiento de las peticiones es tal que el mostrado en cada una de las figuras siguientes, brevemente descrito para cada endpoint.

Para el endpoint `/api/link`, se asocian 3 entradas atendidas por el `link.router.js`:

1. En la ruta `/api/link/`, las peticiones POST serán atendidas por el método `linkRoom`.
2. En la ruta `/api/link/`, las peticiones PUT serán atendidas por el método `unlinkRoom`.
3. En la ruta `/api/link/main`, las peticiones PUT serán atendidas por el método `flagAsMainThing`.

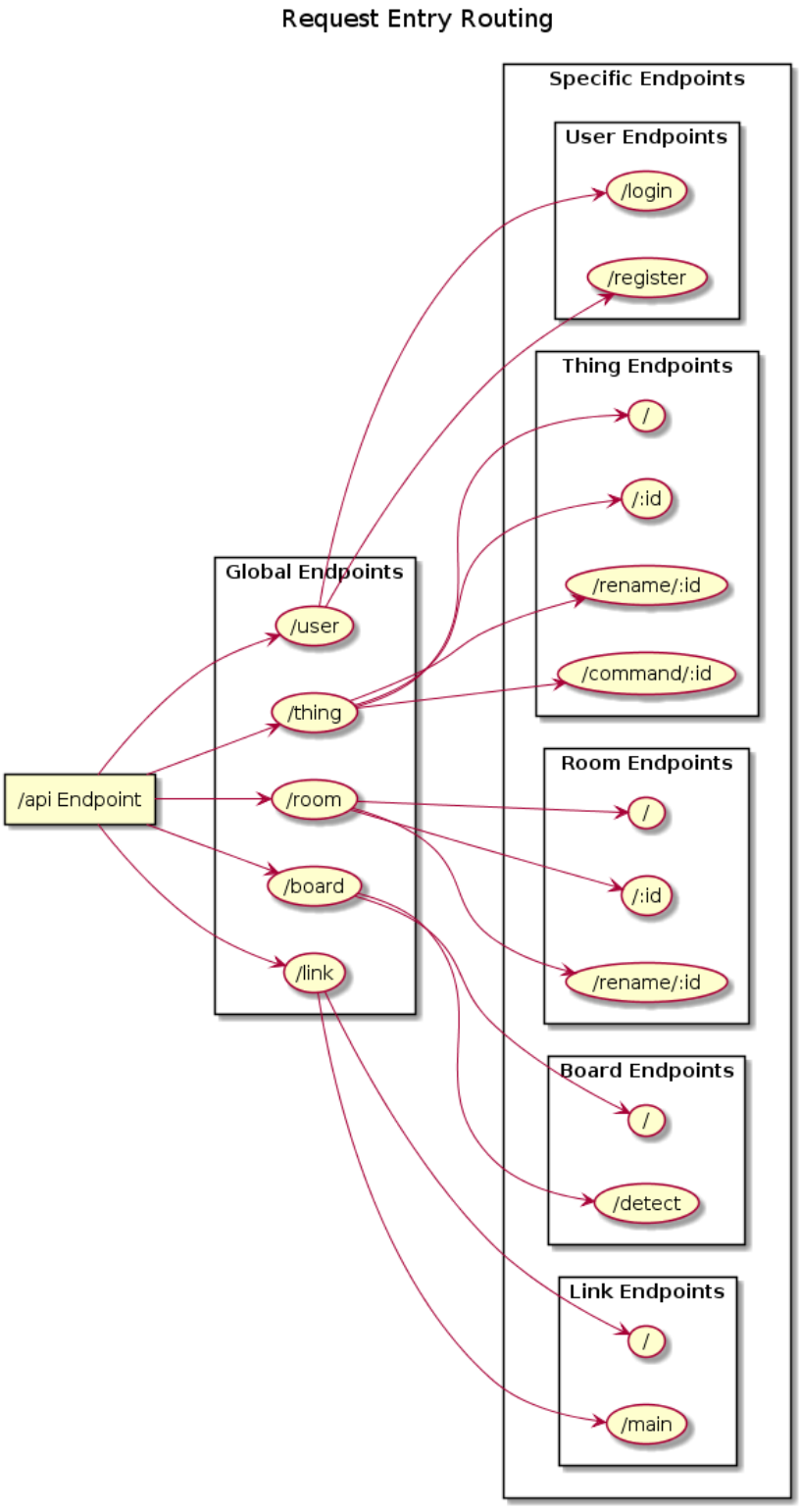

Figura 4.17: *Base API Entry Schema*

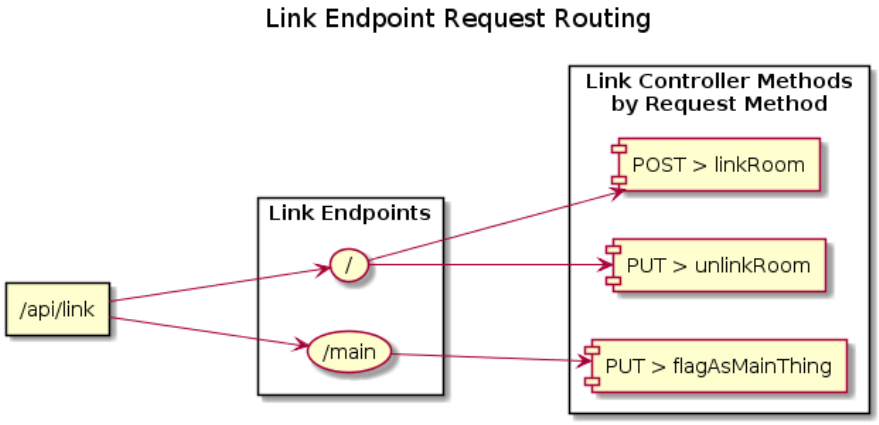

Figura 4.18: *Link API Schema-footnotemark*

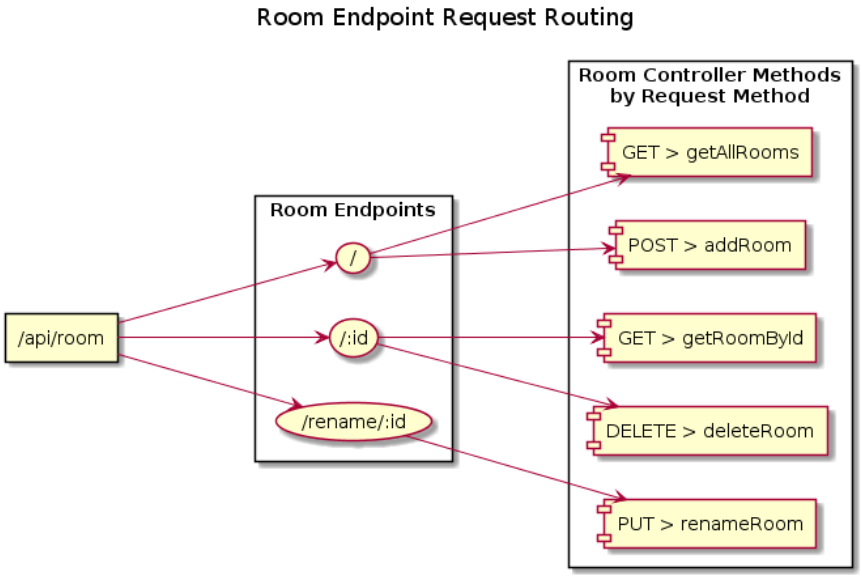

Figura 4.19: *Room API Schema*

Para el endpoint `/api/room`, se asocian 5 entradas atendidas por el `room.router.js`:

1. En la ruta `/api/room/`, las peticiones GET serán atendidas por el método `getAllRooms`.

2. En la ruta `/api/room/`, las peticiones POST serán atendidas por el método `addRoom`.

3. En la ruta `/api/room/:id`, las peticiones GET serán atendidas por el método `getRoomById`.

4. En la ruta `/api/room/:id`, las peticiones DELETE serán atendidas por el método `deleteRoom`.

5. En la ruta `/api/room/rename/:id`, las peticiones PUT serán atendidas por el método `renameRoom`.

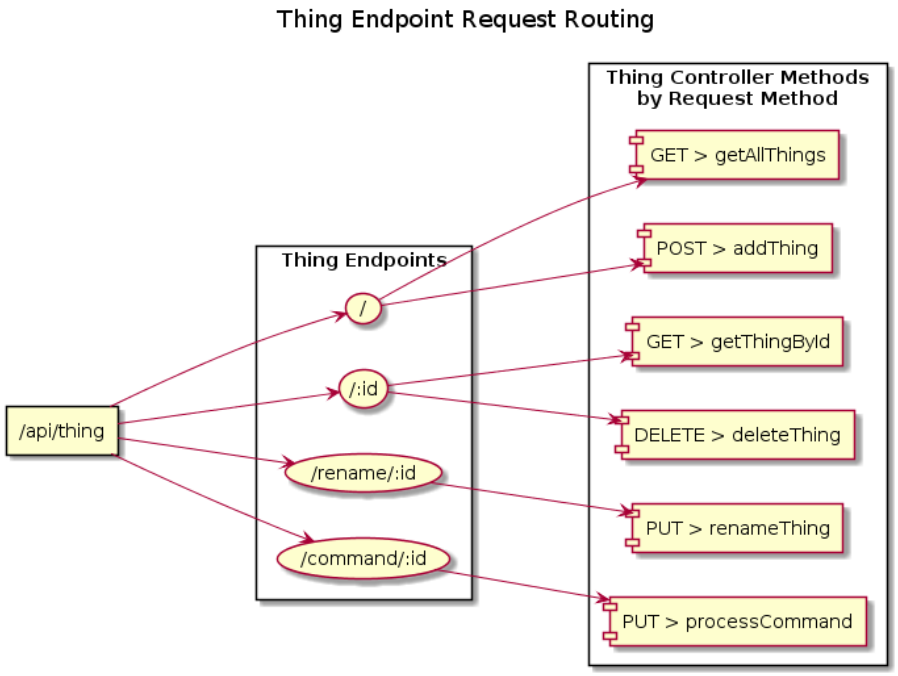

Figura 4.20: *Thing API Schema*

Para el endpoint `/api/thing`, se asocian 6 entradas atendidas por el `thing.router.js`:

1. En la ruta `/api/thing/`, las peticiones GET serán atendidas por el método `getAllThings`.

2. En la ruta `/api/thing/`, las peticiones POST serán atendidas por el método `addThing`.

3. En la ruta `/api/thing/:id`, las peticiones GET serán atendidas por el método `getThingById`.

4. En la ruta `/api/thing/:id`, las peticiones DELETE serán atendidas por el método `deleteThing`.

5. En la ruta `/api/thing/rename/:id`, las peticiones PUT serán atendidas por el método `renameThing`.

6. En la ruta `/api/thing/command/:id`, las peticiones PUT serán atendidas por el método `processCommand`.

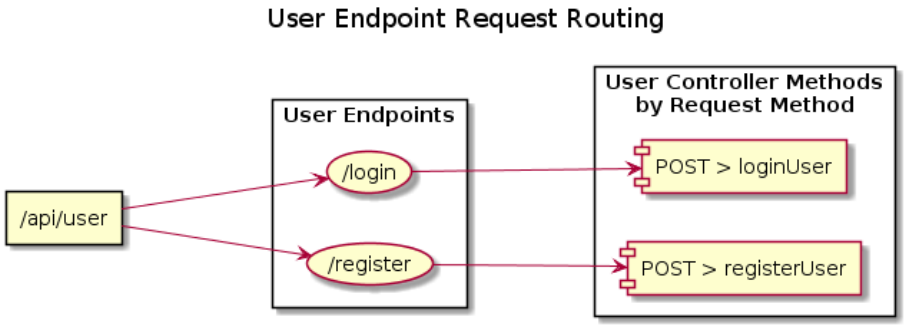

Figura 4.21: *User API Schema*

Para el endpoint `/api/user`, se asocian 2 entradas atendidas por el `user.router.js`:

1. En la ruta `/api/link/login`, las peticiones POST serán atendidas por el método `loginUser`.

2. En la ruta `/api/link/register`, las peticiones POST serán atendidas por el método `registerUser`.

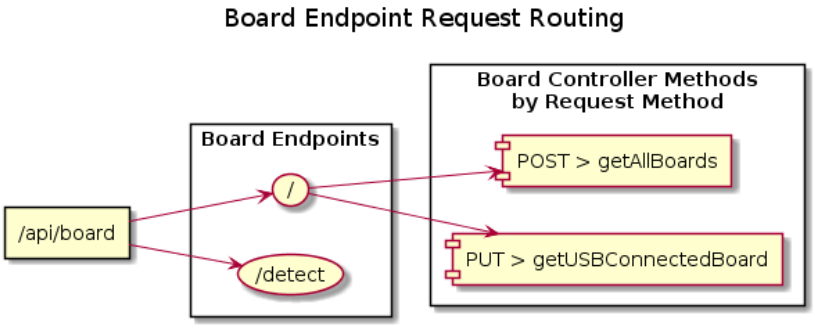

Figura 4.22: *Board API Schema*

Para el endpoint `/api/board`, se asocian 2 entradas atendidas por el `board.router.js`:

1. En la ruta `/api/board/`, las peticiones GET serán atendidas por el método `getAllBoards`.
2. En la ruta `/api/board/detect`, las peticiones GET serán atendidas por el método `getUSBConnectedBoard`.

Hay que hacer mención especial al `passport.middleware.js`. Lo consideramos como un middleware, puesto que asiste al proceso de validación de las peticiones. En este caso se apoya en la librería de npm `passport`, que ayuda a comprobar que la petición en cuestión dispone de un token JWT válido, buscando si existe en la colección Users de la MongoDB un usuario con el mismo id que el contenido en el token. Este middleware aplica en todas las peticiones de la API REST expuesta, salvo en las peticiones de loginUser y registerUser del endpoint /api/user, que no pueden disponer de token por no tener un usuario iniciado. Si la petición no dispone de token, es automáticamente rechazada. Juega un importante papel en el reforzado de seguridad de la aplicación.

Toda petición http que no sea encaminada a un método mediante esta metodología será automáticamente rechazada. Esto permite controlar severamente los caminos por los cuales se ejecutará una lógica en base a una petición http, y por ende constituye una medida de seguridad adicional por el principio básico de conocer y controlar todas las posibles entradas al servidor con un flujo único y específico para cada una de ellas, para minimizar brechas de seguridad.

4.6.4. Diagramas de clases de los Controllers, Helpers y Models

El `core.controller.js` se encarga de lanzar aquellos procesos que deben estar listos tras la inicialización base del servidor NodeJS. Una vez el servidor ha conectado su cliente de Mongo a la instancia de MongoDB, está listo para interactuar con toda la información almacenada anteriormente en la BBDD, y el proceso de `app.js` se encarga de lanzar el

método `setupCoreMonitors` del coreController. Este método lanza dos de sus métodos internos, por un lado `_launchStoredSubscriptions` y posteriormente `_releaseDaemons`, finalizando así la preparación inicial del servidor.

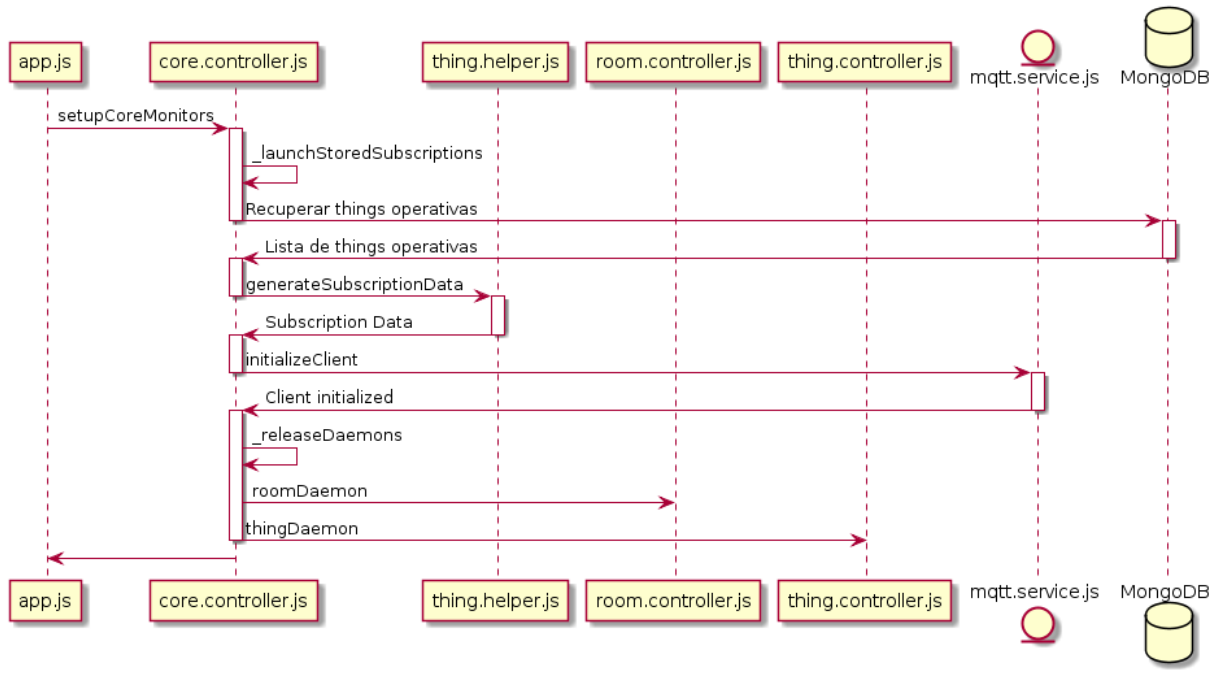

Figura 4.23: *Core Controller sequence diagram*

El método `_launchStoredSubscriptions` se encarga de extraer de la base de datos la lista actual de things y, mediante el método `generateSubscriptionData` del `thing.helper.js`, genera una lista de objetos con información de suscripciones MQTT para cada thing que esté operativa (es decir, vinculadas a una habitación). Tras esto, transfiere dicha lista al `mqtt.service.js` para que inicialice su servicio mediante su método `initializeClient`.

Posteriormente, el método `_releaseDaemons` se encarga de lanzar, a intervalos regulares de 5 segundos, a los *daemons* (denominados así por sus características de funcionamiento) para que paulatinamente realicen sus operaciones regulares, mediante el método `roomDaemon` del `room.controller.js` y el método `thingDaemon` del `thing.controller.js`. Serán explicados en detalle en sus respectivos apartados.

El `room.controller.js` se encarga de, por un lado, correr el `roomDaemon` mencionado previamente, y por otro, exponer sus métodos vinculados a la API REST.

Por un lado, el `roomDaemon` (recordemos que este método es invocado de forma regular cada 5 segundos de operación del servidor) se encarga de calcular la media de las medidas de los sensores DHT11 y MQ135 que haya en cada habitación, y almacenar dicho cálculo en el campo sensorMeasures de cada habitación. Esta operación se lleva a cabo siguiendo los siguientes pasos:

1. Extrae las rooms disponibles en la colección de Rooms de MongoDB, y para cada habitación, lleva a cabo los pasos siguientes pasos.

2. Obtiene la lista de things vinculada a dicha habitación mediante el método `getThingsByRoom` del `thing.controller.js`, y discrimina en dos listas diferentes los things que corresponden a sensores DHT11 y los que corresponden a sensores MQ135.

3. Con la ayuda del `thing.helper.js`, los métodos `getDHT11AvgInfo` y `getMQ135AvgInfo` calculan la información media útil de dichos sensores.

4. Actualiza la room analizada en la MongoDB con esta información almacenada en su campo `sensorMeasures`.

El método `getAllRooms` extrae de la MongoDB la lista de rooms almacenadas, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y la lista de rooms en el body de la respuesta; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `getRoomById` extrae de la MongoDB la room con el campo id igual al parámetro id en la URL de la petición http, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y la room recuperada en el body de la respuesta; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `addRoom` crea un objeto del tipo Room con un id pseudoaleatorio y los campos customName y type coincidentes con los valores del body de la petición http. A continuación,

procede a salvar este objeto en la colección de Rooms de la MongoDB, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y la nueva room creada en el body de la respuesta; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `deleteRoom` elimina de la MongoDB la room con el campo id igual al parámetro id en la URL de la petición http, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y un mensaje de éxito; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `renameRoom` modifica en la MongoDB la room con el campo id igual al parámetro id en la URL de la petición http, sustituyendo el valor de su campo customName por el proporcionado en la petición http, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y un mensaje de éxito; de lo contrario, devuelve un código 500 y un mensaje de error.

El `thing.controller.js` se encarga principalmente de, por un lado, correr el `thingDaemon` mencionado previamente, y por otro, exponer sus métodos vinculados a la API REST. En este caso, por el momento, el método `thingDaemon` no contiene operaciones requeridas pero se mantiene por su potencial utilidad en futuras versiones del proyecto.

El método `getAllThings` extrae de la MongoDB la lista de things almacenadas, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y la lista de things en el body de la respuesta; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `getThingsByRoom` realiza la misma operación que el `getAllThings`, pero sólo extrae de la MongoDB la lista de things almacenadas vinculadas a una room en particular, y devuelve el valor. Este método no atiende a la API REST, sólo responde como utilidad del controller en sí.

El método `getThingById` extrae de la MongoDB la thing con el campo id igual al parámetro id en la URL de la petición http, y si la operación ha tenido éxito, resuelve la

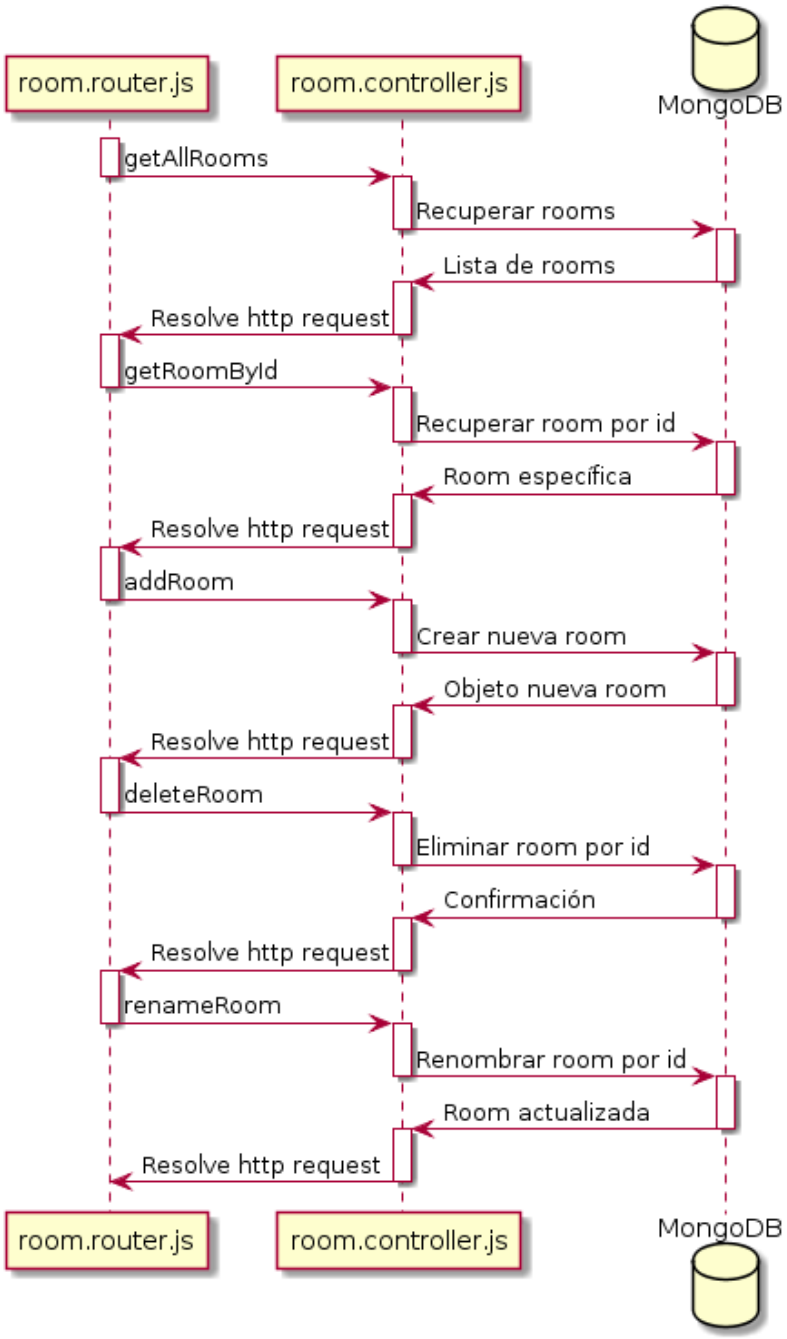

Figura 4.24: *Room Controller sequence diagram*

petición http con un código 200 y la thing recuperada en el body de la respuesta; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `addThing` crea un objeto del tipo Thing con un id pseudoaleatorio y los campos customName, type y model coincidentes con los valores del body de la petición http. Además, en su campo typeProperties, almacena una estructura con datos por defecto dependiendo del type y model especificados (gracias al método `getModelStructure` del `thing.helper.js`). A continuación, procede a salvar este objeto en la colección de Things de la MongoDB, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y la nueva thing creada en el body de la respuesta; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `deleteThing` elimina de la MongoDB la thing con el campo id igual al parámetro id en la URL de la petición http, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y un mensaje de éxito; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `renameThing` modifica en la MongoDB la thing con el campo id igual al parámetro id en la URL de la petición http, sustituyendo el valor de su campo customName por el proporcionado en la petición http, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y un mensaje de éxito; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `processCommand` se encarga de procesar un comando específico para la thing con id igual al parámetro id de la petición http. Primero, obtiene la thing coincidente almacenada en la MongoDB; segundo, obtiene una instancia del controlador específico para dicha thing según su tipo; y por último, ejecuta el método `processRequest` de dicha instancia con los datos de la thing y el comando de la petición. El resultado de este último método expone un flujo de éxito y otro de fracaso. Si la operación ha tenido éxito, resuelve la petición http con un código 200 y un objeto JSON confirmando el éxito del comando solicitado; de lo contrario, devuelve un código 500 y un mensaje de error.

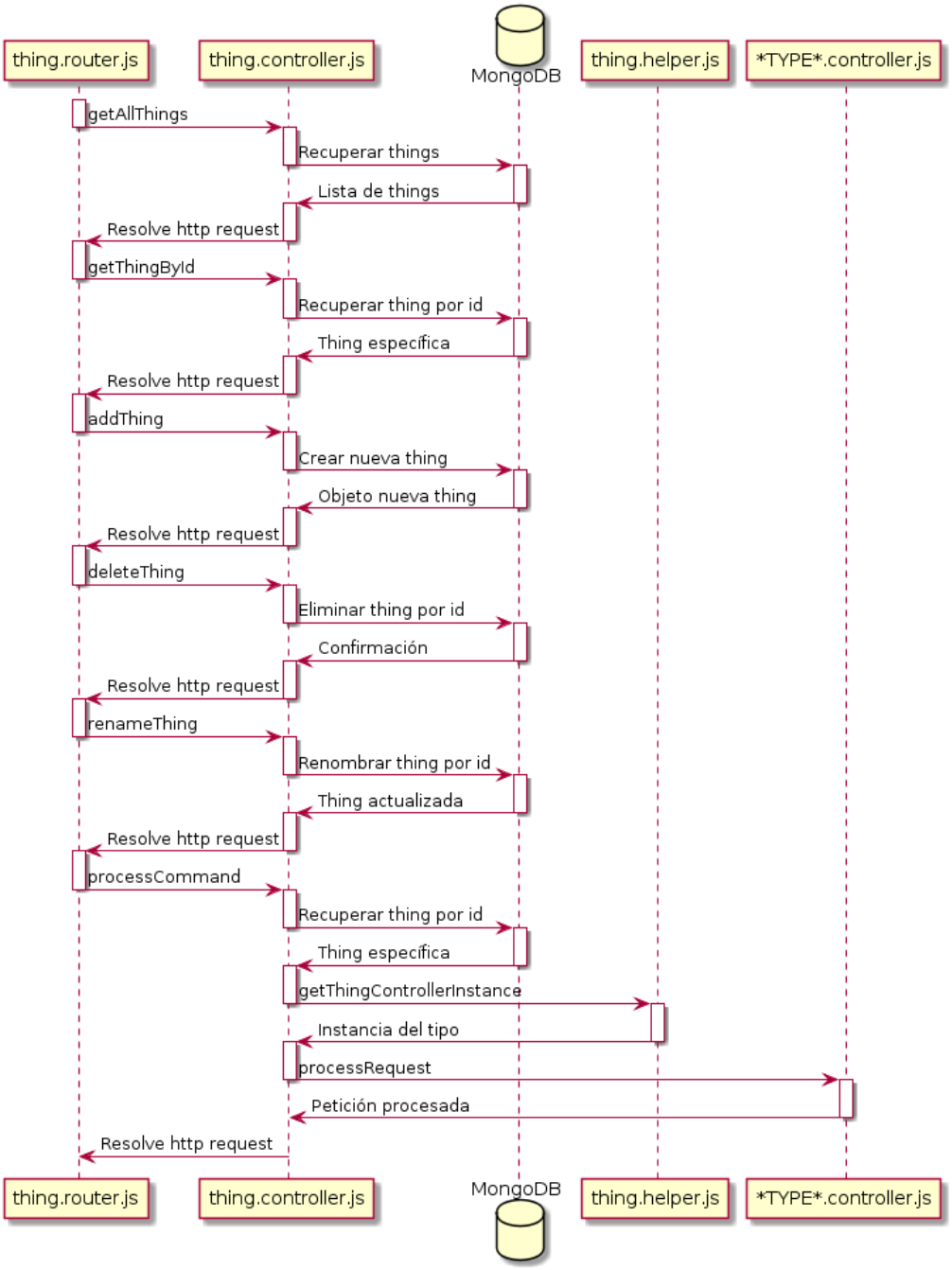

Figura 4.25: *Thing Controller sequence diagram*

El `link.controller.js` se encarga de exponer sus métodos vinculados a la API REST.

El método `linkRoom` se encarga de vincular una thing a una room y de ordenar todo lo necesario para el funcionamiento de dicha thing de forma independiente. Con el fin de evitar repeticiones, se refiere a la sección 5.3.2 para la explicación de este método, pues el contexto proporcionado en dicha sección ofrece una mejor visión del cometido de este método.

El método `unlinkRoom` realiza una operación inversa al `linkRoom`. Así, se encarga de eliminar las referencias mutuas entre la thing de la petición y su room asociada en sus respectivas colecciones Rooms y Things; y cuando esto ha ocurrido con éxito, se encarga de llamar al método `removeSubscription` del `mqtt.service.js` para eliminar tanto la suscripción de *idThing/status* como la de *idThing/answer*; y si la operación ha tenido éxito, resuelve la petición http con un código 200 y un mensaje de éxito; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `flagAsMainThing` se encarga de marcar a la thing de la petición http como principal de su tipo en su room vinculada. Por un lado, actualiza el campo `flaggedAsMain` de la thing (previamente actualiza el mismo campo la anterior thing del mismo tipo marcada como principal antes que ésta, si existe, sustituyéndola así); por otro lado, actualiza en la MongoDB el campo `mainThingsId` de la room asociada en la propiedad pertinente según el tipo de la thing; y si la operación ha tenido éxito, resuelve la petición http con un código 200 y un mensaje de éxito; de lo contrario, devuelve un código 500 y un mensaje de error.

El `board.controller.js` se encarga de exponer sus métodos vinculados a la API REST.

El método `getAllBoards` extrae de la MongoDB la lista de boards almacenadas, y si la operación ha tenido éxito, resuelve la petición http con un código 200 y la lista de boards en el body de la respuesta; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `getUSBConnectedBoard` se encarga de devolver la board conectada en ese momento al puerto USB de la Raspberry Pi. Con el fin de evitar repeticiones, se refiere al

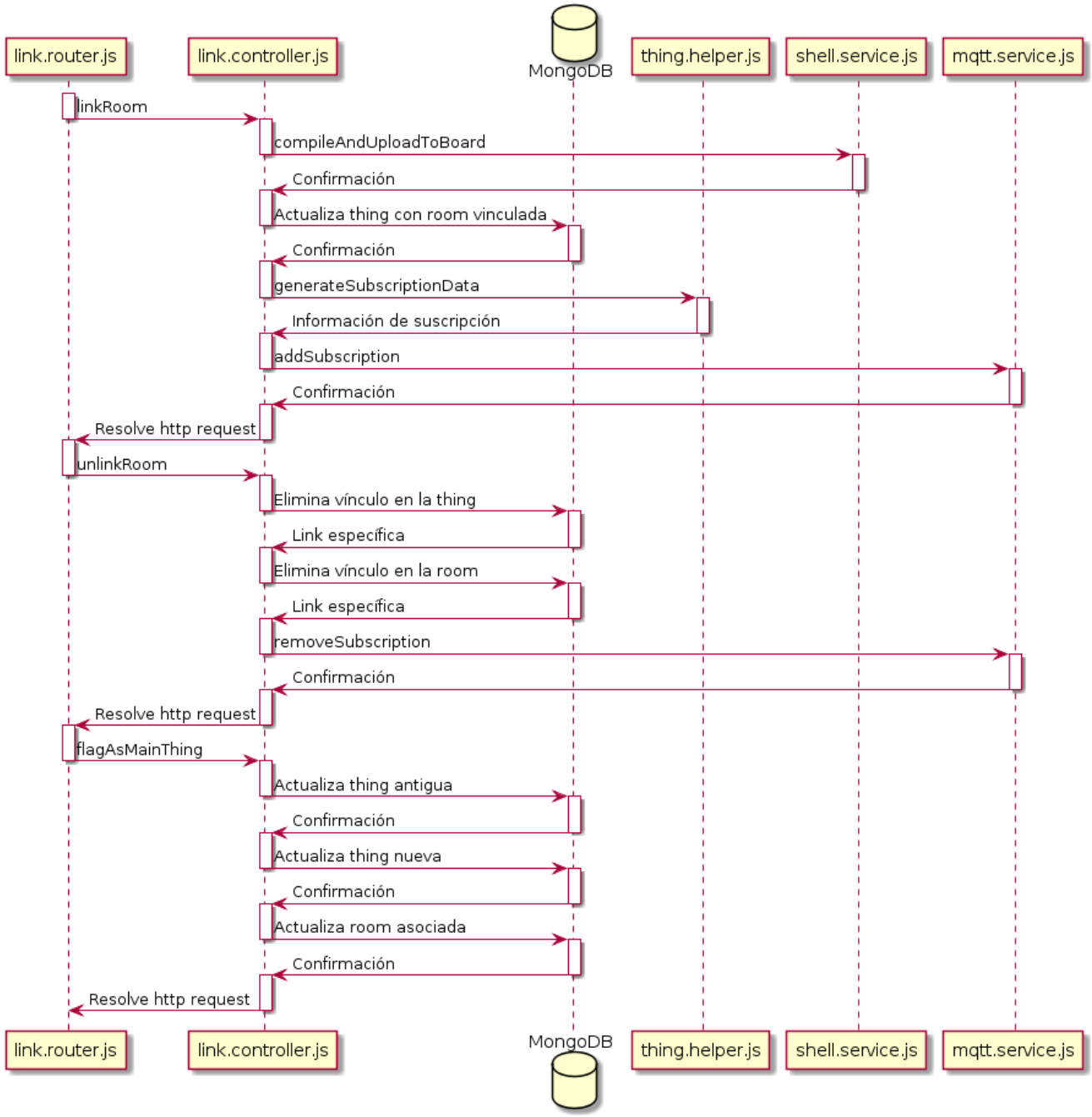

Figura 4.26: *Link Controller sequence diagram*

paso 4 de la sección 5.3.4 para la explicación de este método, pues el contexto proporcionado en dicha sección ofrece una mejor visión del cometido de este método.

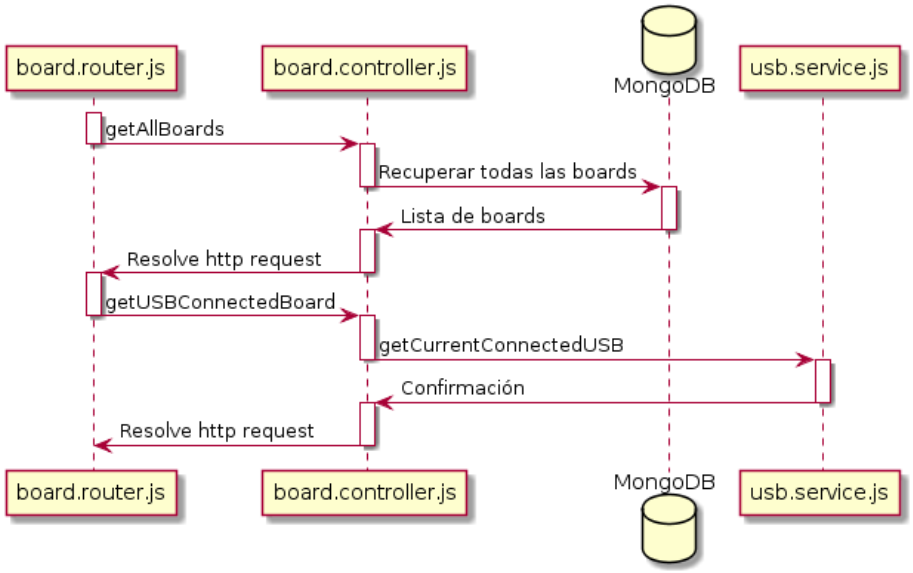

Figura 4.27: *Board Controller sequence diagram*

El `user.controller.js` se encarga de exponer sus métodos vinculados a la API REST.

El método `registerUser` comprueba en la colección Users de la MongoDB si existe un usuario con el email proporcionado en la petición http. Si ya existe, devuelve un código 500 y un mensaje de error; si no existe, crea un nuevo objeto User con las características email y password del body de la petición y lo almacena en la MongoDB. Si la operación ha tenido éxito, resuelve la petición http con un código 200 y un mensaje de éxito; de lo contrario, devuelve un código 500 y un mensaje de error.

El método `loginUser` comprueba en la colección Users de la MongoDB si existe un usuario con el email proporcionado en la petición http. Mediante el método del esquema User `comparePassword`, procede a devolver si el login ocurre con éxito o no, resolviendo la petición http con un código 200 y un objeto JSON con el email y un token válido (creado por el método privado `_createToken`) o bien devolviendo un código 400 y un mensaje de

error.

El método privado `_createToken` hace uso del plugin JWT para crear un token único con el id de usuario y su email, siguiendo la configuración existente en los campos `JWT_SECRET` y `TOKEN_EXPIRE_TIME` del `server.config.js`.

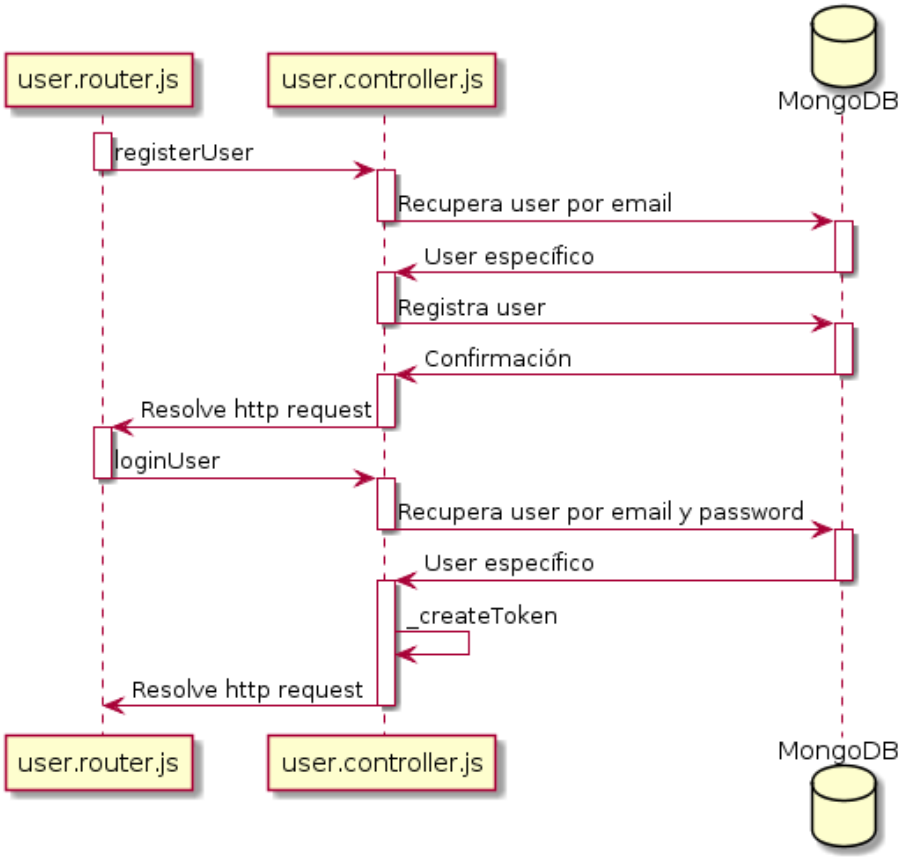

Figura 4.28: *User Controller sequence diagram*

El `light.controller.js`, como controlador directo de acciones de una thing, gestiona las operaciones específicas de una thing de tipo luz por lo que, entre otras, encontramos expuestos los métodos procesar la comunicación MQTT con una luz, además de otros métodos privados auxiliares. Con el fin de evitar repeticiones, no lo explicaremos con más profundidad puesto que lo encontraremos detallado en un contexto real en los pasos del lado servidor del caso de uso de la sección 5.3.

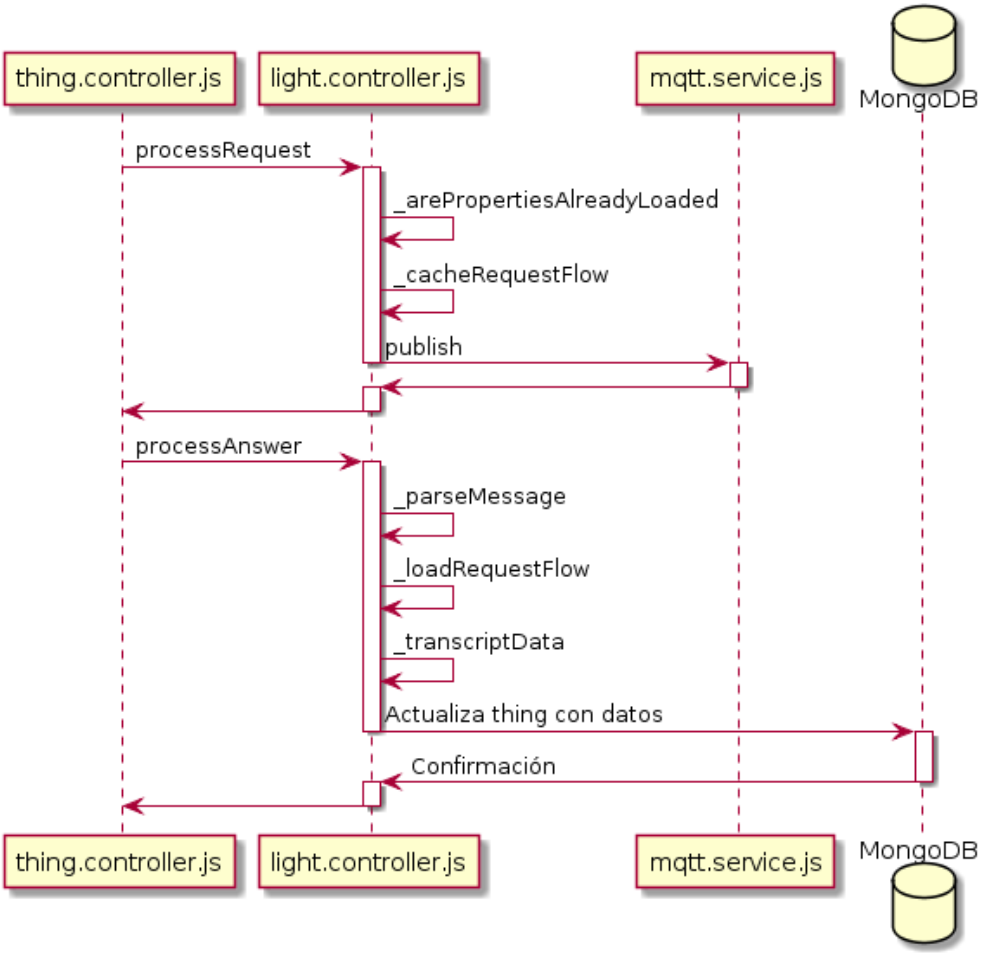

Figura 4.29: *Light Controller sequence diagram*

El `sensor.controller.js`, como controlador directo de acciones de una thing, gestiona las operaciones específicas de una thing de tipo sensor por lo que, entre otras, encontramos expuestos los métodos procesar la comunicación MQTT con una sensor, además de otros métodos privados auxiliares. Con el fin de evitar repeticiones, no lo explicaremos con más profundidad puesto que lo encontraremos detallado en un contexto real en los pasos del lado servidor del caso de uso de la sección 5.4.

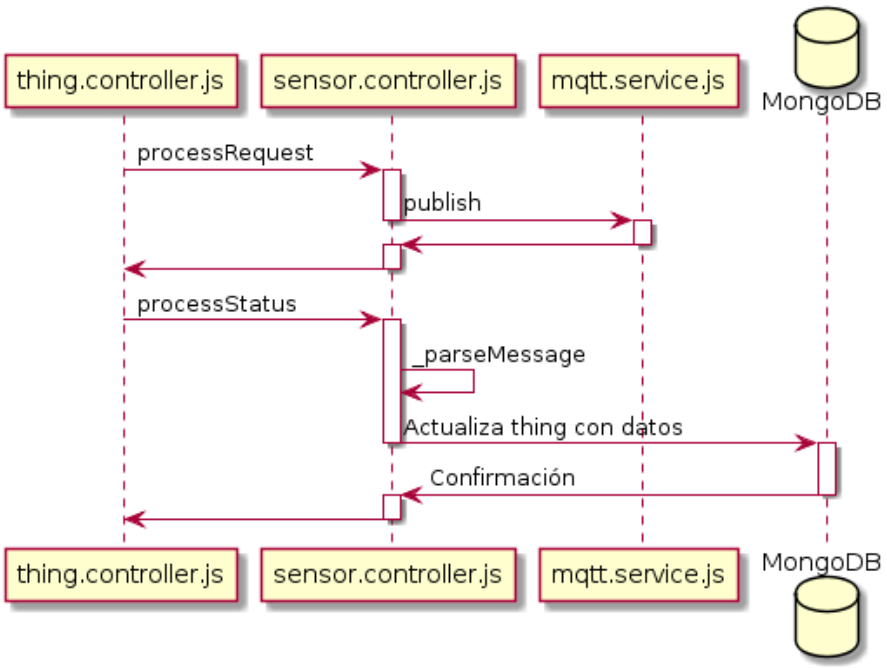

Figura 4.30: *Sensor Controller sequence diagram*

El `thing.helper.js` tiene lugar con el fin de descargar al módulo `thing.controller.js` de código que no está estrictamente vinculado a operaciones contra la base de datos o tratamiento de peticiones o comunicaciones, es decir, de acoger todo aquellos métodos que responden a simples cálculos o utilidades generales relacionadas con los things que son procesos o algoritmos necesarios pero no constituyen el núcleo lógico. Son los siguientes:

1. El método `getThingControllerInstance`, que responde a una necesidad de arquitectura del código, pues se encarga de devolver, a partir de un tipo de thing (luz, sensor,

etc.), una instancia de código de controlador directo de acciones asociado estrictamente a dicho tipo. En la implementación actual de código, este puede ser o bien el `light.controller.js`, o bien el `sensor.controller.js`.

2. El método `generateSubscriptionData`, que a partir de una thing obtiene su instancia controladora correspondiente (con el método descrito anteriormente) y devuelve un objeto con dos campos, `answer` y `status`, donde almacena un objeto en cada uno con la información necesaria para registrar una suscripción MQTT en el servicio MQTT sobre los dos topics necesarios a ser escuchados para la comunicación, el topic `answer` y el `status`, el campo `id` de la thing, y la función de procesamiento asociada a cada topic, bien sea `processAnswer` o `processStatus` de la instancia controladora obtenida.

3. El método `getModelStructure`, que dependiendo del tipo y modelo de una thing, nos devuelve un objeto con la estructura por defecto (y unos valores por defecto) que debe tener el campo typeProperties para almacenar, recuperar y operar los valores acordes a dicho modelo y tipo de thing.

4. El método `getDHT11AvgInfo` sirve para calcular la media de temperatura y humedad de una lista de sensores DHT11 y devolver dichos valores medios.

5. El método `getMQ135AvgInfo`, en la misma línea que el método anterior, a partir de una lista de sensores MQ135, averigua si todos informan de una calidad del aire correcta o si al menos uno de ellos presenta una advertencia de peligro en la calidad del aire, y devuelve este dato junto a la lista de sensores que presentan dicha advertencia. El estado actual de implementación del proyecto no contempla completamente el uso de este sensor, pues aunque toda la estructura de frontend y backend lo soporta, es requerido un voltaje superior al valor con el cual garantizamos que la Raspberry Pi opera correctamente todos sus servicios y comunicaciones.

Los modelos utilizados en el backend se diseñan como modelo para su BBDD de MongoDB, y por ello la creación del modelo es mediante los métodos `Schema` y `model` de la librería de npm `mongoose`. Estos modelos, a excepción del User, siguen el mismo esquema que los utilizados en el frontend, lo que refuerza la mantenibilidad de código y permite una comunicación con menos trabas entre cliente y servidor. El modelo Board, va asociado a la colección de MongoDB 'Boards'; el modelo Room, va asociado a la colección de MongoDB 'Rooms'; el modelo Thing, va asociado a la colección de MongoDB 'Things'; y el modelo User, va asociado a la colección de MongoDB 'Users'.

Particularmente, para el modelo User, posee 2 campos, `email` (identificador único, alineado con el frontend), y `password` para almacenar el password del usuario. Además, se definen unos métodos en relación con su cometido de identificación de usuario para inicios de sesión. Se establece un método que es ejecutado justo antes de almacenar objetos de tipo User en la MongoDB (cuando se registra un nuevo usuario), que gracias a la librería de npm `bcryptjs`, permite aplicar salt y hash sobre el password como medidas adicionales de seguridad. Además, se establece otro método para la comparación de password, en el flujo ejecutado cuando un usuario intenta hacer login. Este método también utiliza la librería de `bcryptjs` para comparar bajo las mismas condiciones de salt y hash.

4.6.5. Diagramas de clases de los Services

El servicio `mqtt.service.js` ofrece la capacidad de interactuar de forma sencilla con otros dispositivos mediante el protocolo MQTT, apoyándose en la librería de npm `mqtt`, mediante los métodos que expone, que explicamos a continuación.

1. El método `initializeClient` permite inicializar un cliente MQTT y conectarlo a un broker mediante una URL. Una vez ha ocurrido, ejecuta el método privado `_loadSubscriptions` con una lista existente de suscripciones necesarias, y lanza el método `_initMessageListener` para iniciar la escucha de mensajes MQTT.

2. El método `stopClient` permite cerrar la conexión con dicho broker MQTT.

3. El método `addSubscription` permite suscribir el cliente a un topic formado por un identificador de thing y un endpoint, y asociar cualquier mensaje recibido en dicho topic a la ejecución de una función referenciada por parámetro.

4. El método `removeSubscription` permite desuscribir el cliente de un topic formado por un identificador de thing y un endpoint y eliminar la vinculación a funciones asociadas en la suscripción.

5. El método `publish` permite publicar un mensaje en un topic formado por un identificador de thing y un endpoint.

Igualmente importante es el flujo de escucha y procesado de mensajes MQTT, mediante los métodos privados `_initMessageListener` y método privado `_processMessage`, que se encargan de procesar cualquier mensaje recibido por MQTT, extraer del topic el identificador de thing y el endpoint y lanzar la función asociada a dicha combinación de suscripción.

El servicio `mongo.db.js` tiene la capacidad de iniciar la conexión entre el proyecto y la BBDD de MongoDB. El trabajo en bruto de la conexión ocurre gracias a la librería de npm `mongoose`.El servicio en cuestión expone el método `_connect` que hace uso de esta librería para, mediante eventos de conexión, establecer la conexión con unos parámetros dados de dónde se encuentra la MongoDB (URL y las opciones necesarias para la conexión, valores que se extraen de los campos `MONGODB_URL` y `MONGODB_OPTIONS` del archivo de configuración `server.config.js`), así como escuchar posibles errores de la conexión y avisar de cuándo la conexión ha tenido éxito, para poder asegurar que a partir de entonces las operaciones contra la MongoDB ocurran con ciertas garantías.

El servicio `shell.service.js` tiene la capacidad de ejecutar scripts de shell. Este servicio expone el método `compileAndUploadToBoard` para generar, compilar y cargar a una placa todos los scripts que necesite dicha placa para operar una thing conectada a ella en

comunicación con el nodo principal. `compileAndUploadToBoard`, para su fin, hace uso del método privado `_execAsync` y detecta cuando el script devuelve por stdout el string *Ended* como señal de finalización. En ese caso, Su método privado `_execAsync` hace el trabajo en bruto gracias al uso de la librería de npm `shelljs`, ejecutando el script de shell que recibe por parámetro. Controla las salidas stdout y stderr para ejecutar una función en caso de resultados devueltos por la salida stdout, o ejecutar otra en caso de errores devueltos por la salida stderr.

El servicio `usb.service.js` es capaz de detectar dispositivos USB conectados al dispositivo sobre el cual corre el proceso de NodeJS.

Este servicio expone el método `initListening` para indicar al servicio que empiece a escuchar los posibles cambios en dispositivos conectados, así como el `stopListening` para indicar al servicio que deje de escuchar dichos cambios. Además, expone el método `getCurrentConnectedUSB` para devolver el dispositivo conectado actualmente. Todo el trabajo en bruto es facilitado por la librería de npm `usb-detection` que interactua con el sistema operativo para ofrecer esta información.

Capítulo 5

Casos de Estudio

La modularidad y movilidad de esta suite domótica permite realizar despliegues rápidos ante un gran abanico de necesidades. Partiendo de un kit básico, compuesto por el nodo principal, unas cuantas placas genéricas equipadas con un modulo esp8266 y unos cuantos sensores y actuadores, se puede prototipar una suite domótica automática personalizada para un situación, evaluar su utilidad y planificar una instalación más profesional si las pruebas obtienen resultados deseados.

5.1. Aplicando la solución en un entorno propuesto

El escenario se desarrolla en una bodega de uso doméstico situado bajo tierra. El acceso a la misma se encuentra dentro de un hogar a través de unas escaleras que bajan hasta la estancia donde una pareja de toneles fermentan vino de cosecha propia de la unidad familiar que reside en la casa. Originalmente este espacio había sido diseñado como almacén y no dispone de medios electrónicos salvo el cableado de iluminación que dispone de un interruptor en la parte superior de las escaleras y una toma de enchufe al final de las mismas.

Existe una serie de condiciones previstas para una buena conservación del vino embotellado en una bodega, entre otras, que la humedad relativa de tu bodega debe estar entre el

65 por ciento y el 75 por ciento. En una bodega demasiado seca pueden llegar a resecarse los tapones, mientras que una excesivamente húmeda presenta el peligro de que se alteren los tapones y las etiquetas debido a la aparición de mohos. La temperatura debe oscilar entre los 10-15C y debe ser, en lo posible, constante. Por otra parte, debe estar bien ventilada pero sin grandes corrientes de aire. Una bodega demasiado fresca ralentiza la evolución del vino y puede ser la causa de que se produzca la precipitación de cristales de tartratos. Por el contrario, en una bodega en la que la temperatura es demasiado alta, el vino tiende a evolucionar más rápidamente y esto puede llegar a traducirse en un envejecimiento prematuro.

Como estrategia, un sensor tomará medidas indefinidamente y los volcara en un fichero/BBDD para recortar los tiempo de respuesta, evitando así esperar a que el sensor haga la toma de medidas en el momento de la solicitud y obteniéndolas en su lugar del último registro que tengamos. Un último problema con el que hay que lidiar es su baja precisión limitada a enteros, por lo que no podemos esperar obtener un dato preciso a la décima de la temperatura y humedad. Esto sin embargo no es un problema real dada la naturaleza del proyecto, ya que no necesitamos un grado de precisión menor a la unidad para tomar acciones o informar al usuario.

5.2. Caso de uso: Vincular un dispositivo registrado a una habitación

El usuario previamente ha registrado en la aplicación una habitación para esta bodega y ha registrado una luz de tipo led. En este momento desea poder vincular esa luz a la bodega, de forma que pueda encenderla y apagarla y observar el estado actual de la luz desde su aplicación. A continuación se describe el proceso por el cual el usuario selecciona las características de la vinculación que va a llevar a cabo (eligiendo la luz, la habitación a la cual vincularla, el modelo de la placa en la cual montará la luz y el pin en el que la conectará) y conecta la placa por USB a la Raspberry (para que el servidor genere,

compile y cargue los scripts necesarios en la placa para que ésta interactúe con la luz y con el nodo principal). Cuando este proceso finaliza, la placa está correctamente vinculada y tanto el backend como el frontend están alineados en información, de forma que el backend será capaz de comunicarse directamente con la luz y retransmitir su estado al frontend, y el frontend podrá controlar su estado mediante comandos lanzados al backend. El usuario podrá observar que la luz ya aparece vinculada estrictamente a la bodega. El diagrama explicativo de este caso de uso es la figura 5.1

5.2.1. Fase 1: Lado aplicación móvil

1. El usuario accede al menú en la pestaña inferior derecha, a su vez al menú *Connect*, localiza la thing que quiere vincular (representada mediante el `thing-block.component`, en este caso la luz de tipo led y procede a deslizarla hacia la izquierda y pulsar en el botón *Link*. Esto desencadena el envío de un evento que escucha el componente padre, `connect.ts`, y lo vincula a la ejecución de su método `linkThing`. Éste se encarga de mostrar el elemento modal que contendrá el asistente de vinculación de la thing a una habitación, cuya lógica estará dispuesta en `wizard-link.modal`. A partir de aquí a cada paso del asistente se le requerirá al usuario información imprescindible para la vinculación, y no podrá avanzar al siguiente paso con el botón *Next* hasta que cumpla lo solicitado en cada paso.

2. En un primer paso, se mostrará al usuario la lista de habitaciones reconocibles mediante su nombre personalizado, para que elija la habitación a la cual quiere vincular esta luz, en este caso, la bodega. Una vez elegido, este dato se preserva hasta el final del asistente.

3. En el segundo paso, se mostrará al usuario la lista de placas disponibles compatibles, con el fin de que elija el modelo de placa sobre el cual montará la luz. En el siguiente paso, se pide al usuario el pin sobre el cual montará la luz. Ambos datos se preservan

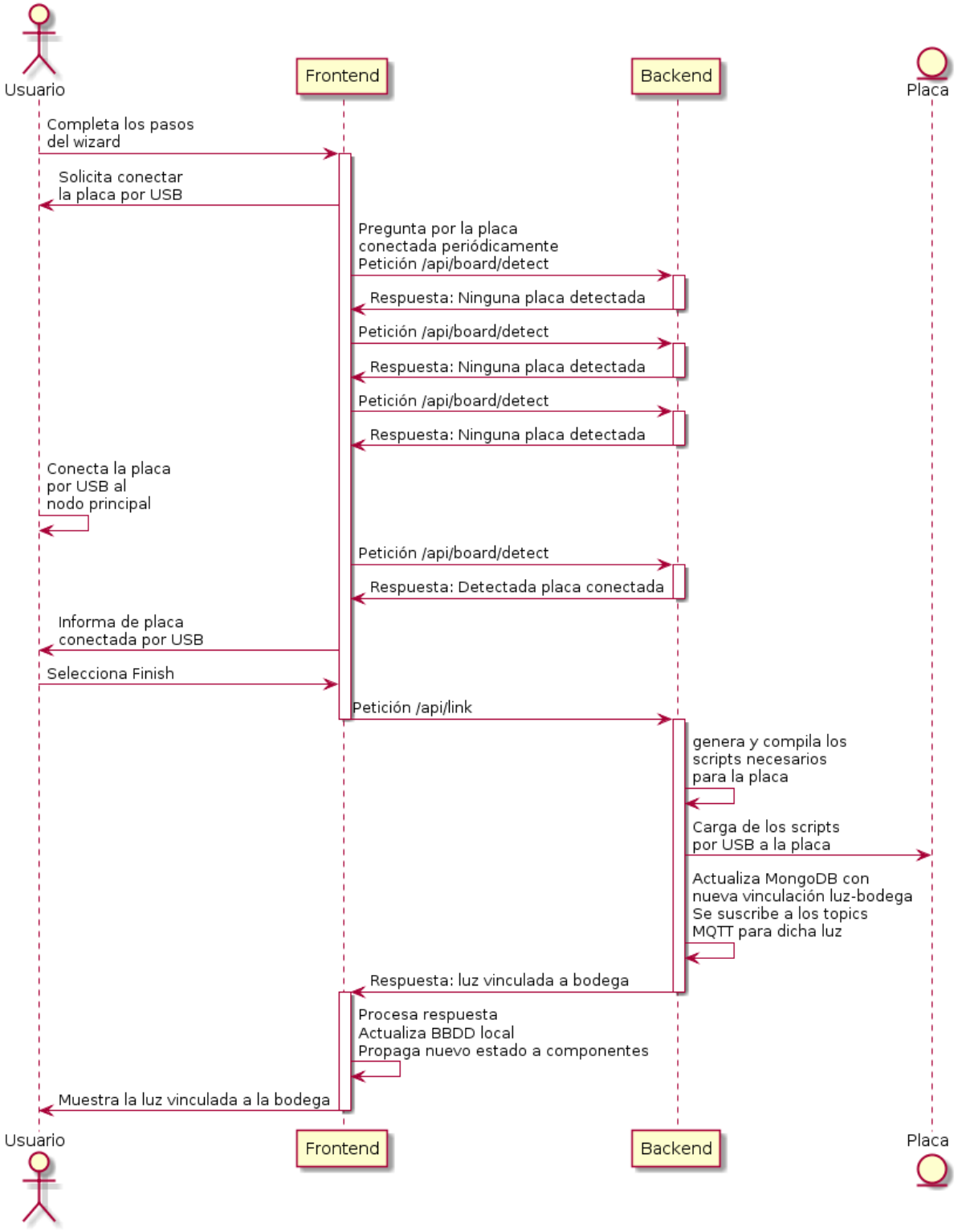

Figura 5.1: *Diagrama de comunicación entre frontend, backend y placa para vincular un dispositivo a una habitación*

hasta el final del asistente.

4. En el último paso, se muestra al usuario un breve mensaje descriptivo de la acción que va a llevarse a cabo con sus elecciones, y se le insta a conectar la placa en la cual ha montado la luz al puerto USB de la Raspberry Pi. Sólo cuando esto ocurra, explicado en más detalle en la Fase Auxiliar de este caso de uso, se permitirá al usuario que lance el proceso de vinculación con el botón *Finish*, el cual ejecuta el método `closeAndFinish`, que a su vez delega el proceso de vinculación en el método `linkRoom` de la `thing.store.ts`, pasando como parámetros la thing vinculada, el id de la habitación, el modelo de placa y el pin asignado.

5. Este método `linkRoom` llamará a su vez (con el id de thing, el modelo de thing, el id de habitación, el modelo de placa y el pin) al método `linkRoom` del `linkProvider` (`api-link.service.ts`), el cual se encarga de comunicar con la api que atiende el servidor en el endpoint `/api/link`. A través del método privado `create`, lanza una petición http de tipo POST, extendida al endpoint `/api/link`, con payload todos los parámetros descritos.

Estos pasos son encadenados mediante promesas (Objetos **Promise** del ECMASCRIPT 6), y tras la respuesta del servidor en la fase 2 de este caso de uso, las promesas se desenrollarán progresivamente hacia atrás en la fase 5, cubriendo los flujos de resolución (si la respuesta es positiva) o los flujos de rechazo (si la respuesta es negativa), detallados en la fase 3.

5.2.2. Fase 2: Lado servidor

1. El router del servidor en `router.js` detecta el endpoint `/api/link` y redirige la petición hacia el `link.router.js`, el cual detecta a su vez la ruta `/` y el tipo POST y redirige la petición a ser controlada por el método `linkRoom` del `link.controller.js`. Obtiene del body de la petición todos los parámetros detallados en el paso 5 de la fase anterior.

2. El método `linkRoom` llamará al método `compileAndUploadToBoard` del `shell.service.js`, que ejecuta el script de shell `sketchgenerator` para una especificación dada, el cual se encarga de obtener el código necesario, compilarlo y cargarlo en la placa conectada por USB a la Raspberry Pi.

3. Si la ejecución del script finaliza con éxito, actualizará la thing en la MongoDB con la nueva habitación vinculada, en el campo `linkedRoomId`, con el método `findOneAndUpdate` del modelo **Thing** (esquema de la DB).

4. Si se actualiza con éxito, entonces llamará al método `generateSubscriptionData` del `thingHelper` (`thing.helper.js`) el cual, a partir del tipo de la thing, obtiene una instancia con la funcionalidad específica necesaria para procesar tanto el status como las respuestas recibidas por MQTT.

5. Con esta información, llamará al `addSubscription` del `mqtt.service.js`, el cual suscribirá al topic *idThing/answer* y a *idThing/status* y guarda en una lista interna de suscripciones las 2 nuevas suscripciones, donde asocia futuros mensajes en dichos topics a los métodos `processAnswer` y `processStatus` respectivamente.

6. Posteriormente, actualizará la habitación en la MongoDB con la nueva thing vinculada, con el método `update` del modelo **Room** (esquema de la DB).

7. Si se actualiza con éxito, desencadena la resolución con éxito de la petición http, enviando de vuelta una respuesta con código http 200 acompañado de un body que contiene un sencillo mensaje descriptivo de éxito.

8. Si alguno de los pasos falla, desencadena la resolución con fracaso de la petición http, enviando de vuelta una respuesta con código http 500 acompañado de un body que contiene un sencillo mensaje descriptivo del error ocurrido.

5.2.3. Fase 3: Lado aplicación móvil

1. La última promesa que aguarda al final de la fase 1 de este caso de uso, tras recibir respuesta exitosa por parte del servidor sobre la petición que realizó (recibe un código 200, y si fuera un 400/500, se procedería a controlar el error con el método heredado `handleError` e iniciar la ejecución del flujo de rechazo), resuelve la promesa, devuelta al flujo a la espera en `linkRoom` de la `thing.store.ts`.

2. El flujo de éxito actualizará el campo `linkedRoomId` de la thing, añadiendo el id de la habitación recién vinculada, y tras esto, se utiliza el método `set` del `ThingDatabaseService` (`thing.service.db.ts`), que se encarga de actualizar la DB local del dispositivo. De la misma forma, tras esto, se acude al método `linkThingToRoom` de la `room.store.ts`, que se encargará en paralelo de igualmente actualizar la habitación correspondiente con la nueva lista de things vinculadas a ella, y actualizar con el método `set` del `RoomDataBaseService` (`room.service.db.ts`) que se encarga de actualizar la DB local del dispositivo.

3. Una vez actualizada la DB local de la aplicación móvil, se llama al método privado `refreshList` de la clase actual, `ThingStore`, con el objetivo de cargar la reciente lista actualizada en la DB y propagar el nuevo valor de esta lista (con sus elementos recién actualizados) a todos aquellos interesados en la aplicación. Esto se consigue mediante la actualización del elemento **Observable** `_currentThingsObservable`, que mediante el patrón de Observer/Subscribe, informa a todos los suscriptores interesados en los cambios de dicho campo (gracias a esto, la lista inicial de things en el menú *Connect* será actualizada con la reciente vinculación de la thing a su habitación). Este mismo proceso ocurre también en paralelo en `RoomStore` pero aplicado a la lista de habitaciones. Por último, se resuelve la promesa, devuelta en este método `linkRoom` al método `closeAndFinish` del `wizard-modal.ts`.

4. Una vez se ejecuta el flujo de éxito en el componente `wizard-link.modal`, se muestra

un breve mensaje de tipo toast informando del éxito en la vinculación mediante el uso del método `showToast` del servicio `toast.service.ts`.

5.2.4. Fase Auxiliar: detección del dispositivo conectado por USB

Precondiciones: -Al iniciar la aplicación servidor, ha sido inicializado el servicio de detección de dispositivos conectados por USB, `usb.service.js` mediante el método `initListening` de este mismo. Así, cada conexión, desconexión o cambio de dispositivo conectado por los puertos USB de la Raspberry Pi será monitorizado.

1. Cuando el usuario alcanza el último paso del asistente de vinculación de dispositivos a habitaciones del `wizard-link.modal.ts`, se lanza el método `startPollingUsbBoard` de la `board.store.ts`, el cual se encarga de lanzar periódicamente el método interno `_retrieveDetectedUsbBoard`, a intervalos de 2 segundos, y se suscribe a los cambios del **Observable** retornado por `detectedBoardObserver` de la `board.store.ts`.

2. El método `startPollingUsbBoard` se encarga de llamar al método `getUsbConnectedBoard` de la `boardProvider` (`api-board.service.ts`), el cual se encarga de comunicar con la api que atiende el servidor en el endpoint `/api/board`. A través del método privado `create`, lanza una petición http de tipo GET, extendida al endpoint `/api/board/detect`.

3. El router del servidor en `router.js` detecta el endpoint `/api/board` y redirige la petición hacia el `board.router.js`, el cual detecta a su vez la ruta `/detect` y el tipo GET y redirige la petición a ser controlada por el método `getUSBConnectedBoard` del `board.controller.js`.

4. Este método solicita al `usb.service.js` si existe una placa conectada en ese mismo momento mediante el método `getCurrentConnectedUSB`. Si el servicio había registrado un dispositivo conectado, lo retornará, desencadenando la resolución con éxito de la petición http, enviando de vuelta una respuesta con código http 200 acompañado de un body que un objeto **Board** (según esquema de la MongoDB) con la información

de la placa. Si no había ningún dispositivo registrado, desencadenará la resolución de fallo de la petición http, enviando de vuelta una respuesta con código http 500 acompañado de un breve mensaje descriptivo de la ausencia de dispositivos conectados por USB.

5. Tras recibir respuesta exitosa por parte del servidor sobre la petición que realizó (recibe un código 200, y si fuera un 400/500, se procedería a controlar el error con el método heredado `handleError` e iniciar la ejecución del flujo de rechazo), resuelve la promesa del paso 2, recuperando el flujo a la espera en `getUsbConnectedBoard` de la `board.store.ts`.

6. Éste, dependiendo de la respuesta, actualizará el valor `_detectedUsbBoard` existente (si es positiva, añadirá un objeto con la información disponible de dicha board; si la respuesta es negativa, añadirá un objeto vacío). Mediante el patrón Observer/Subscribe, avisará al suscriptor del `wizard-link.modal.ts`, para que, al detectar cuando existe una board conectada por USB, habilita el botón *Finish*.

5.3. Caso de uso: Encender un dispositivo luminoso de tipo led

El usuario ha registrado en la aplicación una habitación para esta bodega, ha registrado una luz de tipo led y la ha vinculado a dicha bodega mediante lo explicado en el caso de uso anterior. En este momento, el usuario decide encender dicha luz. El proceso descrito a continuación detalla cómo el frontend responde al input del usuario que desplaza el interruptor de luz en el componente visual, comunicando el comando de encendido al servidor mediante una petición http, cómo el backend procesa el comando para entender qué acción debe llevar a cabo y envía por MQTT el mensaje adecuado a la luz adecuada, y cómo se produce la respuesta desde la placa, el backend reacciona a la respuesta por MQTT de la placa, actualiza el estado de la luz en su base de datos y devuelve la petición http al fron-

tend, el cual procesa la respuesta desplazando el botón e iluminando el componente visual de la luz. El diagrama explicativo de este caso de uso se muestra en la Figura 5.3.

5.3.1. Fase 1: Lado aplicación móvil

1. Desde uno de los componentes visuales, bien sea el `room-preview.component.ts`, o sea el `light-component.ts`, tras interacción del usuario con el toggle (interruptor) de esta luz, se desencadena el método `toggleMainLight` o `toggleLight` respectivamente, los cuáles ejecutan un algoritmo similar que por un lado deshabilita el interruptor para el usuario durante el breve lapso de procesamiento que ocurre hasta que se reciba una respuesta en la fase 5 de este caso de uso, y por otro lado llama al método `turnOn` de la `thing.store.ts` con la información de la thing en cuestión (nuestra luz) como parámetro.

2. Éste llamará al método privado `setProperty` pasándole la thing y un objeto de la clase **CommandRequestModel** con parámetros de comando para encender.

3. `setProperty` delega en el método `setProperty` del `thingProvider` (`api-thing.service.ts`), el servicio que se encarga de comunicar con la api que atiende el servidor en el endpoint `/api/thing`. A través del método privado `update`, lanza una petición http de tipo PUT, extendida al endpoint `/api/thing/command/:id`, con el id de la thing como parámetro id de la url, y el comando como payload.

   Estos pasos son encadenados mediante promesas (Objetos **Promise** del ECMASCRIPT 6), y tras la respuesta del servidor en la fase 4 de este caso de uso, las promesas se desenrollarán progresivamente hacia atrás en la fase 5, cubriendo los flujos de resolución (si la respuesta es positiva) o los flujos de rechazo (si la respuesta es negativa).

5.3.2. Fase 2: Lado servidor

1. El router del servidor en `router.js` detecta el endpoint `/api/thing` y redirige la petición hacia el `thing.router.js`, el cual detecta a su vez la ruta `/command/:id` y

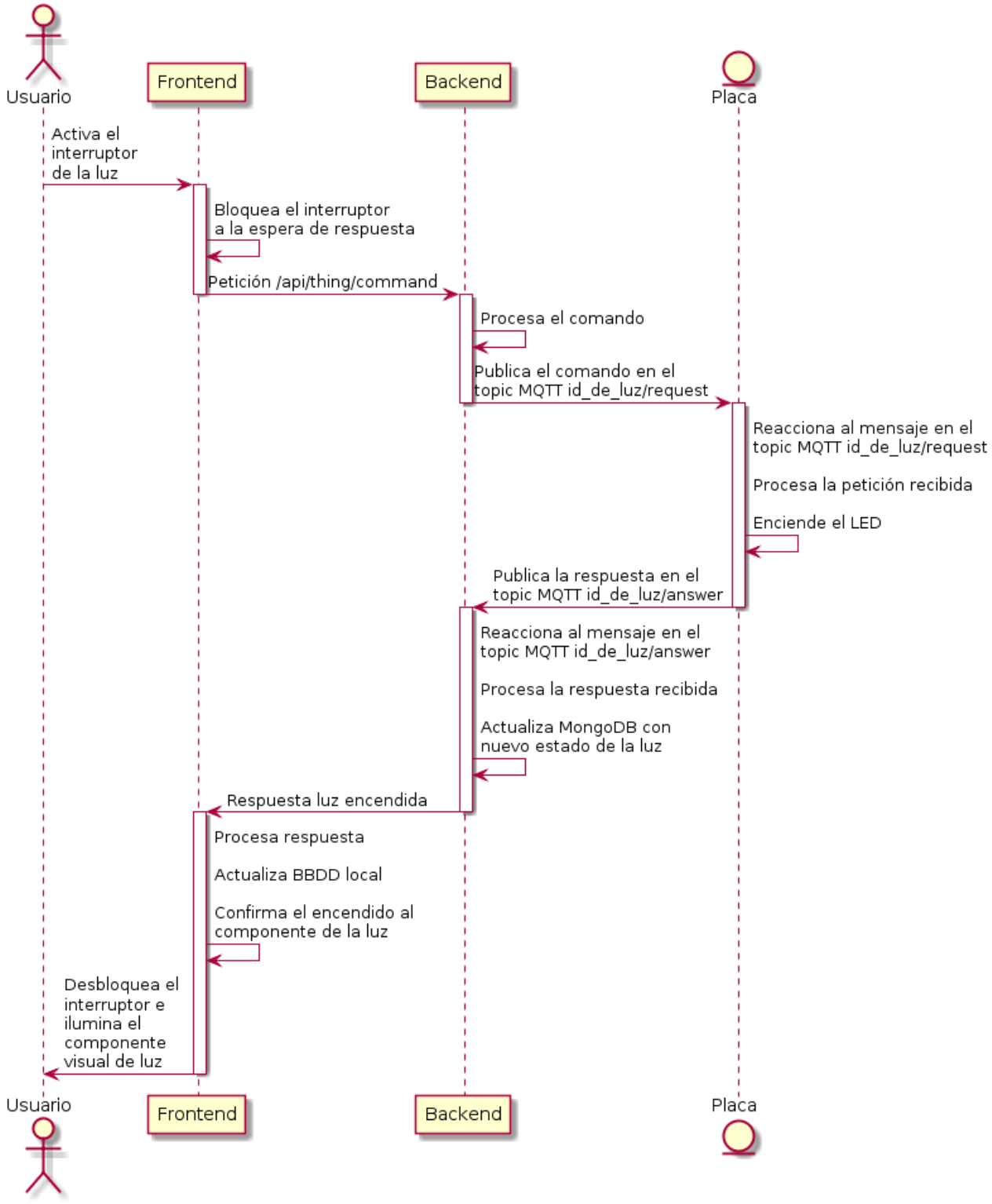

Figura 5.2: *Diagrama de comunicación entre frontend, backend y placa para encender un dispositivo led*

el tipo PUT y redirige la petición a ser controlada por el método `processCommand` del `thing.controller.js`. Obtiene del body de la petición el comando solicitado.

2. Éste realiza una acceso a la MongoDB mediante el método `findOne` del modelo **Thing** (esquema de la DB), para obtener los datos almacenados de la thing en cuestión mediante el id extraído de la url.

3. Si existe y logra los datos, llamará al método `getThingControllerInstance` del `thingHelper` el cual, a partir del tipo de la thing (tipo light), obtiene una instancia con la funcionalidad específica necesaria para procesar un comando y enviarlo al dispositivo. En este caso, el método será el `processRequest` del `light.controller.js`.

4. En el `light.controller.js`, se utiliza el método interno `_arePropertiesAlreadyLoaded` para comprobar, a partir del comando y los datos almacenados de la thing, si los cambios de propiedades que indica el comando implican un cambio real en el estado actual (almacenado) del dispositivo. Esto se hace para evitar comandos en paralelo que puedan dar lugar a comportamientos inesperados.

5. Si el comando debe efectivamente enviarse, se procede a extraer el string equivalente del comando, y registrar una nueva entrada en la lista interna de comandos pendientes mediante el método interno `_cacheRequestFlow`. Este sistema permite guardar la traza de un comando dirigido hacia una thing en particular con unas propiedades en particular con el objetivo de asociar a dicha traza un flujo de éxito y otro de fracaso, que se ejecutarán respectivamente cuando en la fase 4 de este caso de uso se recoja la respuesta de éxito o fracaso del comando en el dispositivo.

6. Se llama entonces al método `publish` del `mqtt.service.js`, que se encargará a su vez de publicar el mensaje final en el topic compuesto *thingId/request*.

5.3.3. Fase 3: Lado dispositivo

1. La placa se encuentra en un estado de continua escucha del topic `request` para su propio identificador. El proceso de resolución de mensajes entrantes por parte del servidor, recibe un string con un formato JSON `"{\"powerStatus\": \"power_on\"}"` el cual es parseado en una estructura de array que permite identificar el contenido del comando, en este caso, la solicitud de alterar el estado de la señal saliente en su pin configurado para la iluminación del led.

2. Alterado el estado de la señal de iluminación y la variable global que permite consultar el estado del led, se procede a publicar la respuesta de la operación en el topic `answer` de su propio identificador con un string JSON con el contenido `"{\"powerStatus\": \"power_on\"}"`.

5.3.4. Fase 4: Lado servidor

1. Se recibe un mensaje MQTT con el string `"{\"powerStatus\": \"power_on\"}"` en el topic *thingId/answer*. La escucha de mensajes MQTT en el `mqtt.service.js` ejecuta un proceso que extrae el id de thing *thingId* y la categoría *answer* y llama al método interno `_processMessage`, con esos datos y el string.

2. Este método delega en `_findSubscriptionIndex` la necesidad de localizar la información de la suscripción correspondiente en la lista interna y, si existe, ejecuta el callback asociado a esta thing para la categoría *answer*, en este caso, se trata del método `processAnswer` del `light.controller.js`.

3. En el `light.controller.js`, se intenta parsear el mensaje mediante el método `_parseMessage`, y se espera que devuelva un objeto JSON.

4. Si lo parsea con éxito, se procede a llamar al método interno `_loadRequestFlow`. Este método accede a la lista de comandos pendientes registrados, explicados en el paso 5 de la fase 2 de este caso de uso. De dicha lista, obtiene el flujo de éxito correspondiente

al comando descrito en la respuesta parseada, garantizando así que se resuelve la traza almacenada original.

5. Dicho flujo de éxito desencadena la resolución con éxito de la petición http original, enviando de vuelta una respuesta con código http 200 acompañado de un body que contiene por un lado el comando original solicitado en el campo `commandRequest` y por otro lado un campo `answer` con un mensaje descriptivo de éxito (cumpliendo así el formato esperado por el cliente).

5.3.5. Fase 5: Lado aplicación móvil

1. La última promesa que aguarda al final de la fase 1 de este caso de uso, tras recibir respuesta exitosa por parte del servidor sobre la petición que realizó (recibe un código 200, y si fuera un 400/500, se procedería a controlar el error con el método heredado `handleError` e iniciar la ejecución del flujo de rechazo), parsea el cuerpo como un **CommandAnswerModel** y resuelve la promesa, pasando dicha información al flujo a la espera en `setProperty` de la `thing.store.ts`.

2. El flujo de éxito se encarga de llamar al método privado `updateThingWithCommand` con la thing y la respuesta, el cual se encarga de discernir el tipo de la thing para llamar a un método que procese correctamente el comando respuesta, en este caso `updateLightWithCommand`, que se encarga de parsear el contenido del objeto de clase **CommandAnswerModel**, de forma que dependiendo de los valores que contenga, modificará con un contenido u otro las diferentes propiedades de la thing, en este caso el campo `powerStatus` de la propiedad `typeProperties`. Esta modificación ocurre con el objetivo de, justo después, hacer uso del método `set` del `ThingDatabaseService` (`thing.service.db.ts`), que se encarga de actualizar la DB local del dispositivo.

3. Una vez actualizada la DB local de la aplicación móvil, se llama al método privado `refreshList` de la clase actual, `ThingStore`, con el objetivo de cargar la reciente

lista actualizada en la DB y propagar el nuevo valor de esta lista (con sus elementos recien actualizados) a todos aquellos interesados en la aplicación. Esto se consigue mediante la actualización del elemento **Observable** `_currentThingsObservable`, que mediante el patrón de Observer/Subscribe, informa a todos los suscriptores interesados en los cambios de dicho campo. Por último, se resuelve la promesa, devuelta en este método `setProperty`, a su vez en el método `turnOn`, y a su vez devuelta al método `toggleMainLight` o el método `toggleLight` de `room-preview.component.ts` o de `light.component.ts` respectivamente.

4. Una vez se ejecuta el flujo de éxito en el componente visual, se rehabilita el interruptor por el usuario, y se actualizan los valores internos que marcan la parte visual del componente, desplazando el interruptor hasta la posición de encendido, informando así al usuario de que su acción se desencadenó con éxito.

5.4. Caso de uso: Procesamiento de información de un sensor

El usuario ha registrado en la aplicación una habitación para esta bodega, ha registrado un sensor de tipo DHT11 (sensor de temperatura y humedad) y lo ha vinculado a dicha bodega mediante lo explicado en el caso de uso de la sección 5.3. En este momento, el usuario decide averiguar la temperatura y humedad de la bodega. El proceso descrito a continuación detalla cómo la placa publica mediante MQTT la información de su sensor DHT11, cómo el backend escucha y procesa el mensaje de estado del sensor, actualiza la base de datos del backend almacenando la nueva información, y cómo el frontend pide periódicamente información al backend sobre las habitaciones y las things conectadas, proporcionando al usuario una información en tiempo real de los dispositivos, y por ende, de la temperatura y humedad de la bodega. El diagrama explicativo de este caso de uso es la Figura 5.4, y para la fase auxiliar, la Figura 5.4.3.

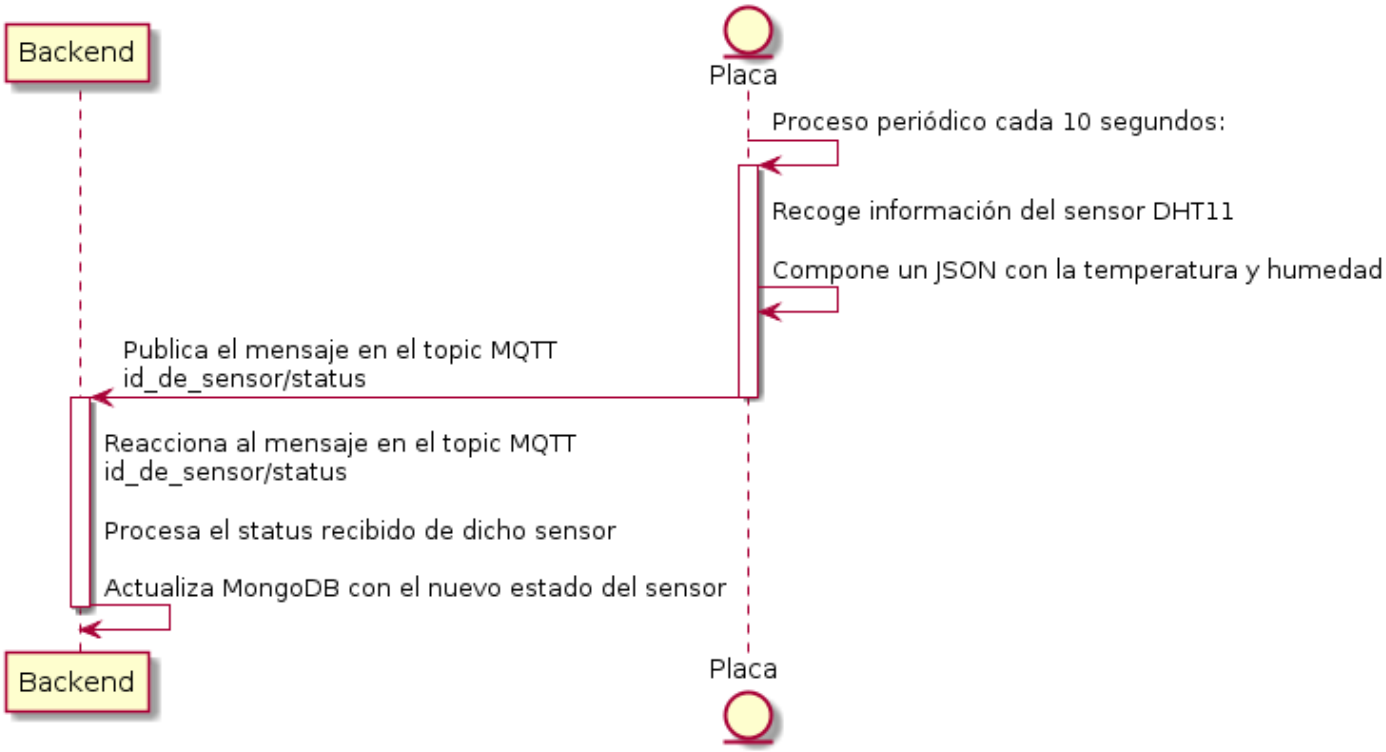

Figura 5.3: *Diagrama de comunicación entre backend y sensor para comunicar su status*

5.4.1. Fase 1: Lado dispositivo

El dispositivo se encuentra en un estado permanente de publicación de estado, iterando cada 10 segundos un bucle que recoge la información del sensor de temperatura y humedad, procesando la respuesta en un string con formato JSON con el siguiente contenido `"{\"temperature\": 28, \"humidity\": 53}"` y publicándolo en el topic *thingId/status*. Véase que los contenidos del JSON aquí reflejados son meramente orientativos a la respuesta esperada.

5.4.2. Fase 2: Lado servidor

Precondiciones: el cliente MQTT del servidor ha sido inicializado mediante el método `initializeClient` del `mqtt.service.js`, el cual conecta a la dirección **mqtt://localhost**, y si tiene éxito, mediante el método `_loadSubscriptions` recupera cada thing que haya almacenada en la MongoDB (y que esté vinculada a una habitación) y se suscribe a los topics `answer` y `status` de cada una. Cuando acaba, prepara mediante el método interno `_initMessageListener` una escucha a todos los posibles mensajes MQTT que pueda recibir el cliente conectado, y dicha escucha, al recibir un mensaje, lanza el proceso descrito a continuación.

1. Se recibe un mensaje MQTT con el string `"{\"temperature\": 28, \"humidity\": 53}"` en el topic *thingId/status*. La escucha de mensajes MQTT en el `mqtt.service.js` ejecuta un proceso que extrae el id de thing *thingId* y la categoría *status* y llama al método interno `_processMessage`, con esos datos y el string.

2. Este método delega en `_findSubscriptionIndex` la necesidad de localizar la información de la suscripción correspondiente en la lista interna y, si existe, ejecuta el callback asociado a esta thing para la categoría *status*, en este caso, se trata del método `processStatus` del `sensor.controller.js`.

3. En el `sensor.controller.js`, se intenta parsear el mensaje mediante el método

`_parseMessage`, y se espera que devuelva un objeto JSON.

4. Si lo devuelve con éxito, actualizará la thing en la MongoDB con el objeto JSON parseado en el campo `sensorMeasures` de la propiedad `typeProperties`, con el método update del modelo **Thing** (esquema de la MongoDB).

5.4.3. Fases paralelas: cálculo de datos generales de la habitación asociada y recuperación de datos en tiempo real

La aplicación backend corre el proceso `roomDaemon` del módulo `room.controller.js` explicado en la sección 4.6.4, mediante el cual evalúa a intervalos regulares de 5 segundos la información general de una habitación en particular con los cálculos medios de las medidas de los sensores asociados a dicha habitación, y almacena dicha información de la habitación en la colección de Rooms, asociando la información a la habitación.

De la misma forma y al mismo tiempo, la aplicación frontend corre el proceso `_synchronizeData` de la `room.store.ts` y el proceso `_synchronizeData` de la `thing.store.ts`, mediante los cuales recupera del backend a intervalos regulares de 5 segundos la información actual de rooms y things. Los procesos están descritos en la sección 4.5.5. En pro de la claridad, describiremos el proceso relativo al caso de uso actual hablando únicamente del flujo para las rooms, pero tenga en cuenta el lector que a la vez ocurre el proceso paralelo para las things, con sus métodos homónimos en sus respectivos módulos homónimos.

1. La lógica asociada a la vista que esté usando el usuario en ese momento es la que se suscribe al observable `roomsChange` de la `room.store.ts`, siguiendo el patrón Observer/Subscribe. Puede darse el caso tanto en la vista global `home.ts`, en la que observamos los datos resumidos de todas las habitaciones registradas y será la descrita en esta subsección, como en la vista particular de la room `room.ts`, para el caso actual la bodega, en la que observaremos los datos generales de la habitación así como los datos individuales de cada dispositivo asociado.

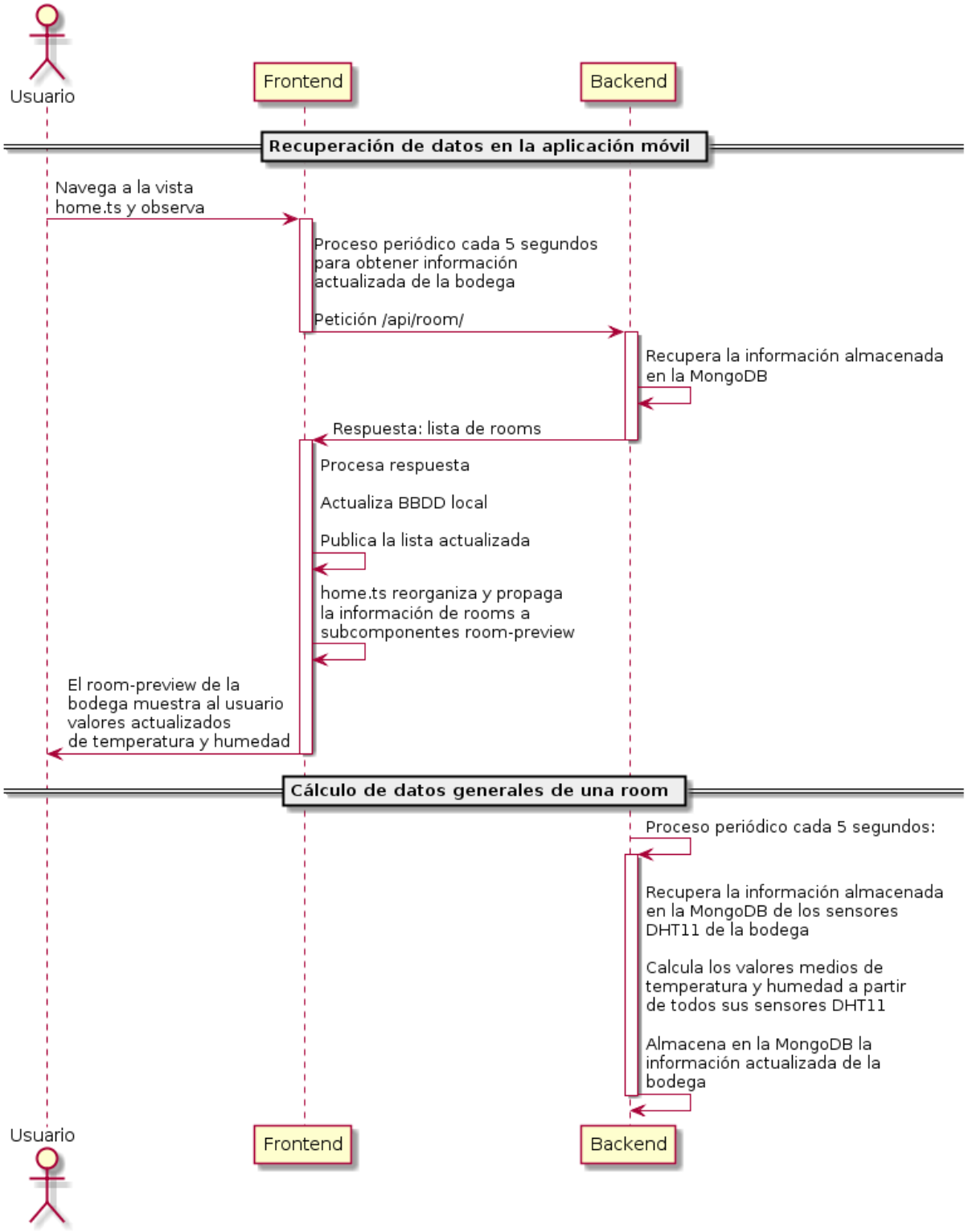

Figura 5.4: *Diagrama de procesos en frontend y backend para actualizar información de sensores en una room y mostrarlo al usuario en tiempo real*

2. Según se ha descrito al principio de esta subsección, cuando la aplicación recupera los datos y progresa la ejecución del método `_synchronizeData` de la `room.store.ts`, se notifica al suscriptor del paso anterior la lista actualizada de rooms, de forma que se ejecuta el desencadenante del suscriptor creado en el constructor de `home.ts`. Dicho desencadenante lanza el método privado `indexThingsInRooms` para encajar la información actualizada de cada thing en cada room asociada, dentro de una lista de rooms.

3. Mediante el paradigma de componentes, el template `home.html` instancia un componente `room.preview.component` por cada room de la lista anterior, y le transfiere la información de dicha room. El template de dicho componente encuentra los valores generales de temperatura y humedad dentro del campo `sensorMeasures` del objeto `room` y los muestra finalmente como valores numéricos para información del usuario.

Capítulo 6

Conclusión y mejoras

El estudio requerido para la creación de una suite domótica ha representado un reto dada la extensión de opciones tecnológicas, estándares establecidos y protocolo de arquitectura y comunicación disponibles. Aun no habiendo podido experimentar con pruebas reales todos los estudios planteados en el estado del arte, puede confirmarse una selección adecuada de tecnologías para abordar un proyecto de IoT que enfrente los problemas característicos de una suite domótica.

El uso del protocolo MQTT ha demostrado ser una de las opciones más sencillas para abordar la comunicación y transferencia de mensajes entre nodos y gateway. Posee una baja carga de consumo energético en las placas microcontroladoras y facilita la gestión de identidades de dichas placas en la red domótica gracias al planteamiento de publicación y subscripción de topics. Además, debido al uso de cadenas de caracteres en formato JSON, es sencillo manejar las respuestas y peticiones del servidor de la aplicación. Las primeras pruebas realizadas sobre el protocolo HTTP demostraron un mayor consumo energético y de proceso como resultado de ejecutar un servidor web en cada placa individualmente. Esto implicaba que las respuestas por parte de las placas pudieran demorarse hasta 5 segundos cuando el servidor solicitaba información a las mismas. Estas placas además, estuvieron conectadas a pequeñas baterías de 5v con capacidad para 3500mA y fueron agotadas en menos de 4 horas. En las pruebas sucesivas, el código de las placas utilizo el protocolo

MQTT reduciendo los tiempos de respuesta de las placas a las peticiones del servidora un máximo de 2 segundos y la dirección de las mismas baterías se amplio hasta 8 horas. Ambos protocolos están basados en el estándar TCP y eso permite una comunicación más confiable entre dispositivos, permitiendo incluso plantear el uso de técnicas de cifrado para la comunicación.

El uso de la tecnología WI-FI es un acercamiento válido en un hogar de pocas habitaciones, pero la atenuación de señal es evidente en las paredes de las habitaciones, y si el gateway se encuentra ubicado a una distancia superior a tres habitaciones de un nodo, la señal se ve lo suficientemente debilitada para que se produzcan pérdidas de paquetes o caídas de la conexión. Esta restricción debe ser tenida en cuenta en despliegues, ya que la estructura de los hogares es muy variable, y cada elemento arquitectónico del edificio posee un impacto diferente en la atenuación de las señales, por ejemplo, un muro de carga posee un mayor grosor que un muro divisorio, y la degradación de la señal es más notable en el primero. Igualmente, los suelos que separan los pisos son aún más restrictivos que los propios muros, en los pisos dúplex o chalet de varias plantas, si la señal tiene su origen en una habitación en el extremo opuesto de la habitación a la que queremos llegar, estando ésta además en una planta distinta, generalmente es poco probable que la señal llegue con suficiente calidad como para operar con normalidad. Otras tecnologías inalámbricas como IPv6 over Low power Wireless Personal Area Networks (6lowpan) podrían resolver estos casos con mejor desempeño, ya que poseen un mayor alcance. También puede valorarse explotar capacidades adicionales del chip `ESP8266` integrado en las placas utilizadas en este prototipo para realizar un despliegue en malla, que permitiría a los nodos actuar como enrutadores de otros nodos, ampliando así el alcance total de la red en la distribución del hogar. Esta opción, sin embargo, debe ser sopesada en función del coste energético que implica en los nodos.

Sobre la cuestión de consumo energético de los nodos y el propio gateway, es evidente que un nodo cuya duración de batería es inferior a medio día, no es práctico ni siquiera como prototipo. Hay que considerar en base a las medidas tomadas mediante un polímetro, que el uso normal de WI-FI en los nodos de este proyecto consumen entre 50mA a 150mA, esto manteniendo una conexión constante en la red. Las placas microcontroladoras que en su SoC instalan el chip `ESP8266`, disponen de funciones de hibernación que permiten apagar la mayoría de los elementos del nodo reduciendo de manera notable dicho consumo. Una estrategia valida en un despliegue de IoT puede aprovechar esta capacidad reduciendo el numero de comunicaciones entre el nodo y el gateway, haciendo que la aplicación del servidor requiera de tomas de medidas programadas cada cierto tiempo en vez de manera constante o bajo demanda del usuario. en todo caso, este problema responde a si la alimentación de las placas es suministrada por baterías o se encuentran conectadas a la red eléctrica del hogar. Sólo en el primer caso las cuestiones de consumo son realmente relevantes.

A continuación se describen en distintas secciones como abordar los problemas mencionados a partir del desarrollo ya realizado en el prototipo, y mejoras de diseño e implementación que han quedado fuera del alcance original del proyecto.

6.1. Refuerzos de la seguridad en la comunicación inalámbrica

La librería pubSubClient [O'L] utilizada en la generación de sketch esta implementada para ser simple de usar. Sin embargo, tiene limitaciones en lo que respecta al cifrado de conexiones cliente-servidor mediante el protocolo MQTT. No posee soporte para conexiones cifradas `SSL/TLS`. Existen librerías ya disponibles de este protocolo que pueden aportar estas funciones, o ampliar la utilizada en este proyecto apoyándose en otras librerías que complementan la seguridad de las conexiones. Una aproximación deseable en el proyecto consta de reforzar dos puntos concretos de las conexiones inalámbricas de la suite domótica.

Primero, establecer conexiones con la red WI-FI mediante librerías mas robustas como `WiFiClientSecure` que admite cifrados asimétricos [Ard19] con firma y verificación.

El segundo paso requiere que el broker de información del nodo principal resuelva las conexiones mediante el protocolo TLS en lugar de TCP simple. Esta estrategia debe estudiarse antes de su implementación, ya que este protocolo, siendo más seguro, implica una mayor sobrecarga en la red con los handshake TLS de conexiones de corta duración, como ocurre con los actuadores, cuya comunicación se limita a recibir un comando y reportar un resultado. También debe considerarse la mayor carga de proceso que se aplica a los dispositivos y el propio nodo principal. En una red con una veintena de dispositivos que establecen comunicaciones de corta duración con contenidos de mensajes de apenas un par de decenas de caracteres, la sobrecarga de comunicación puede ser detrimental hasta el punto de hacer que la suite funcione anormalmente lenta. Por ello, la primera capa de cifrado en la red inalámbrica será una ampliación obligatoria y la segunda capa debe ser estudiada antes de su aplicación.

6.2. Mejora en la cobertura inalámbrica de la suite domótica

En términos de conectividad inalámbrica, la distancia de despliegue física de dispositivos está limitada al rango de emisión del adaptador WI-FI que actúa como router. En este proyecto, el nodo principal de la suite domótica es dicho proveedor de red y los dispositivos se conectan dentro de su rango operativo. Las atenuaciones de señal son particularmente notables dentro de una casa, ya que existen múltiples elementos arquitectónicos como paredes, suelos y columnas, así como la presencia de otros dispositivos con funciones de conectividad inalámbrica. De hecho, una suite domótica, en carácter general siempre va a contar con estas dificultades como norma. Por ello debe plantearse una estrategia que establezca una zona de conexión con una señal estable y de un alcance a diez metros, a partir del cual se

hace notable la caída de las señales WI-FI emitidas por un punto de acceso, tal y como se recoge en un articulo de la universidad de Stanford sobre modelado de atenuaciones de señales WI-FI, apartado 3, "Modeling Attenuation Indoors-[F+05].

Aunque existen repetidores de señales que pueden ampliar el rango efectivo de una wireless local area network (WLAN), en realidad, se están generando nuevos puntos de acceso con nombres distintos, y esto complica mucho el despliegue de nuevos dispositivos en la red de la suite domótica, teniendo que preveer qué nombre de WLAN le corresponde a un dispositivo que, por no estar a un alcance óptimo del nodo principal, debe conectarse a un repetidor concreto que le suministre mejor red. Incluso si se resuelve este problema, solo estaríamos consumiendo más tomas de corriente para alimentar dichos repetidores. Pero los chips `esp8266` utilizados en este proyecto disponen de la capacidad de ser desplegados con una configuración de WI-FI en malla. Conocido como WI-FI Mesh, se trata de una arquitectura de comunicación inalámbrica donde cada dispositivo actúa como enrutador en la red, siendo cada uno un punto de acceso para otros nodos, lo que permite desplegar dispositivos mas allá del alcance del router del nodo principal.

6.3. Actualización de código OTA en dispositivos

El despliegue de nuevos dispositivos en la red domótica mediante una primera conexión USB, ha demostrado evitar errores característicos de sincronización inalámbrica como ocurre en dispositivos de las marcas comerciales estudiadas en el estado del arte. Esto se debe principalmente a que no es necesario una etapa de sincronización y verificación por parte del usuario, ya que al conectar una placa al gateway, se cargan todos los credenciales necesarios para las sucesivas conexiones. Sin embargo, existen ciertos escenarios donde poder alterar el código de las placas sin necesidad de volver a conectarlas al gateway con una conexión alámbrica se puede convertir en un requisito primario. A tal fin, las placas que integran módulos `ESP8266` disponen en sus librerías de capacidad de subir nuevo código mediante

conexión WI-FI. Esto es un gran avance respecto a las placas de Arduino originales que requerían de la ampliación física de hardware mediante los denominados `shields` los cuales incluso no ofrecían esta capacidad en sus productos comerciales iniciales e implicaban la creación de nuevos dispositivos de hardware que solucionaban esta problemática. Puede observarse un ejemplo de implementación de esta idea en un trabajo de tesis realizado en la UPC nombrado "Design of an Arduino shield for ota programming-citecampa2013design.

Un escenario previsto que podría beneficiarse son los cambios en la configuración WI-FI del gateway. Si al verse comprometida la contraseña de acceso inalámbrica a la red domótica, el usuario quisiera cambiar este parámetro, esto implicaría conectar todas y cada una de las placas ya desplegadas mediante USB al gateway para subir la nueva contraseña al código. Sin embargo, con la estrategia inalámbrica, podría ejecutarse un procedimiento que aplicase este cambio en todas las placas. Esta misma estrategia es aplicable a la hora de mejorar el código de las placas mediante refactorizaciones o actualizaciones de las librerías, lo cual permite mantener un sistema domótico actualizado y más fácil de mantener.

6.4. Ampliando el nodo principal con interfaz táctil

El prototipo está pensado para operarse mediante un smartphone, pero la interfaz visual puede ser replicada de manera local en una pantalla táctil conectada directamente al gateway. La estrategia más sencilla para aprovechar el trabajo ya desarrollado implica otro servidor local de NodeJS sobre el mismo nodo, que serviría en una IP local la aplicación cliente, de forma que se pudiera acceder a la aplicación a través del navegador del propio nodo principal y se podría mostrar al usuario dicho navegador mediante una pantalla conectada. De esta manera puede tenerse una sesión adicional de cliente conectada al servidor de la aplicación domótica y no depender del smartphone dentro del hogar.

Un añadido adicional de esta estrategia permite hacer del gateway un dispositivo con un

diseño elegante a la par que funcional, que puede instalarse en un lugar visible del hogar sin que su presencia se sienta como la de un prototipo de pruebas. Tomemos por ejemplo, una aportación del usuario `James Clough` de la comunidad de Thingiverse[Clo] donde expone una carcasa con atalaje para una pantalla de 7"que puede replicarse con una impresora 3D.

Figura 6.1: *Carcasa de raspberryPi con Panel diseñado por James Clough*

El modelo actual del frontend y el backend han sido diseñados con buenas capas de abstracción con el fin de poder seguir creciendo tanto horizontal como verticalmente. El sistema actual por el cual la aplicación móvil es informado por el servidor del estado actual de las habitaciones y los dispositivos permitiría añadir una funcionalidad de notificaciones inteligentes, mediante las cuáles enviaría un aviso a la aplicación móvil cuando el servidor detectase ciertos eventos o patrones, parametrizables por el usuario. Así, cuando el servidor detectara un incremento lento pero gradual de la temperatura en una habitación durante el día, notificaría al dispositivo móvil dicho evento y sugeriría al usuario que decidiera si bajar las persianas o encender el aire acondicionado de dicha habitación.

6.5. Desplegando informes de evolución histórica de medidas capturadas

En la misma línea que la sección anterior, se puede añadir una funcionalidad adicional que a la hora de calcular los datos generales de cada habitación guarde un estado particular de dicha habitación en otra colección con el objetivo de establecer un historial de mediciones. De esta manera, se podría no sólo controlar la evolución de cada una de esas mediciones y detectar posibles anomalías mostrándole al usuario detalladas gráficas, por ejemplo, de temperatura y humedad para una habitación en particular, sino además utilizar dicha información como base para la detección de patrones por el asistente inteligente descrito anteriormente.

6.6. Valoración de impacto ambiental

Se ha determinado depositar todos los componentes dañados durante el proceso de estudio de hardware, y aquellos que en el futuro dejen de funcionar en los puntos limpios de recogida establecidos por la comunidad de Madrid más cercanos. El uso del material de impresión `PLA` en las cubiertas de componentes como la Raspberry Pi no esta basado en petroleo y está considerado de bajo impacto medioambiental en su uso, sin embargo, no es un material reciclable como el caso del `ABS` que es procesado para crear pellets con el fin de crear nuevos componentes plásticos, pero, al tratarse de un material fabricado en base a plantas como el maíz, es biodegradable en un rango de tiempo de unos meses si se aplica correctamente la técnica de compostaje. Por ello los restos de impresión, piezas no válidas o defectuosas en el proceso de creación serán depositadas en los cubos de basura orgánicos.

Glosario

APP Abreviatura de aplicación, generalmente orientada a aplicaciones que se ejecutan en dispositivos móviles.

broker nodo que establece las comunicaciones de una red de suscriptores y publicadores de información para realizar notificaciones inteligentes.

dataset una colección de datos generalmente tabulada..

docker Contenedores de software para el despliegue de aplicaciones de forma automática.

framework conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática.

gateway dispositivo y/o software que actúa como punto de conexión entre dispositivos inteligentes y sensores.

handshake TLS proceso donde dos actores comunicandose en un medio intercambian mensajes para reconocerse, verificarse, establecer los algoritmos de cifrado que usarán y acordar las claves de sesión..

hash funciones que cifran una entrada.

Machine Learning método de análisis de datos que automatiza la construcción de modelos analíticos.

middleware software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él..

Plug and Play enchufar, conectar y usar.

salt bits aleatorios que se usan como una de las entradas en una función derivadora de claves.

script archivo de procesamiento por lotes.

sketch Un sketch es el nombre que Arduino usa para un programa. Es el código que se carga y ejecuta en una placa Arduino o derivada..

smartPhone teléfono inteligente.

wizard interfaz de usuario que faicilita un proceso de instalación o gestión asistida.

Bibliografía

[Ama19a] Amazon. Condiciones de uso de alexa. https://www.amazon.es/gp/help/customer/display.html?nodeId=201809740, 2019.

[Ama19b] Amazon. Condiciones de uso y venta en españa. https://www.amazon.es/gp/help/customer/display.html?nodeId=200545940, 2019.

[Ama19c] Amazon Ltd. Alexa, dispositivos Echo y tu privacidad. https://www.amazon.es/gp/help/customer/display.html/?nodeId=GA7E98TJFEJLYSFR, 2019.

[Ard19] Arduino. Client secure class for esp8266. https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/client-secure-class.html, 2019.

[BCG07] Raffaele Bruno, Marco Conti, and Enrico Gregori. Throughput analysis and measurements in ieee 802.11 wlans with tcp and udp traffic flows. *IEEE Transactions on Mobile Computing*, 7(2):171–186, 2007.

[Bra89] Robert Braden. Requirements for internet hosts-communication layers. *Computer*, 1989.

[Bru18] Jorge Jarne Brun. *Smart Home usando IoT y Chatbots*. ucm, 2018.

[Clo] James Clough. https://www.thingiverse.com/thing:1646204. (Esquemática de raspberry Pi Modelo 3B).

[Com] Wikipedia Comunity. Protocolo de comunicación x10. https://es.wikipedia.org/wiki/X10. Accessed: 2019-04-07.

[DRA19] Drasko DRASKOVIC. https://mainflux.readthedocs.io/en/latest/, 2019. (Drasko DRASKOVIC in the role of BDFL).

[Esp19] Xiaomi España. Política de privacidad de xiaomi: Xiaomi españa: Mi.com - xiaomi españa. https://www.mi.com/es/about/privacy, 2019.

[F+05] Daniel B Faria et al. Modeling signal attenuation in ieee 802.11 wireless lans-vol. 1. *Computer Science Department, Stanford University*, 1, 2005.

[FOUa] FIWARE FOUNDATION. What is fiware. https://www.fiware.org/developers/. Accessed: 2019-04-09.

[Foub] Raspberry Pi Foundation. https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3b_1p2_reduced.pdf. (Esquemática de raspberry Pi Modelo 3B).

[Fouc] Raspberry Pi Foundation. *Securing your Raspberry Pi*. Accessed: 2015-10-12.

[Fun18] Raspberry Pi Fundation. *Securing your Raspberry Pi*, 2018.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[Goo19a] Google. Más información sobre la seguridad y la privacidad de los datos en los dispositivos compatibles con el asistente - ayuda de google nest. https://support.google.com/googlehome/answer/7072285?hl=es&ref_topic=7173611, 2019.

[Goo19b] Google. Política de privacidad – privacidad y condiciones. https://policies.google.com/privacy?hl=es, 2019.

[j.m14] j.m.sánchez. Protección de datos multa a google con 900.000 euros por infracciones graves. https://www.abc.es/tecnologia/redes/20131219/abci-google-multa-aepd-201312191217.html, Jan 2014.

[KKSV17] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[MKHC07] Gabriel Montenegro, Nandakishore Kushalnagar, Jonathan Hui, and David Culler. Rfc 4944: Transmission of ipv6 packets over ieee 802.15. 4 networks. *Request for Comments*, 2007.

[ML08] Xin Ma and Wei Luo. The analysis of 6lowpan technology. In *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, volume 1, pages 963–966. IEEE, 2008.

[Mur] Antonio Merchán Murillo. https://www.abogacia.es/2018/08/27/internet-of-things-y-proteccion-de-datos/. (Internet of Things y Protección de Datos).

[ndcdEMS] S.M.E. Instituto nacional de ciberseguridad de España M.P SA. https://www.incibe-cert.es/blog/seguridad-comunicaciones-zigbee. (Pruebas de laboratorio para estandar Zigbee).

[Noe15] Noel. Comparativa raspberry pi, odroid, banana pi y matrix arm. https://lignux.com/comparativa-raspberry-pi-odroid-banana-pi-y-matrix-arm/, 2015.

[oC] openHAB Community. What is fiware. https://www.openhab.org/docs/installation/rasppi.html. Accessed: 2019-04-10.

[O'L] Nick O'Leary. *Arduino Client for MQTT*.

[SD16] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[SF19] Ed. S. Farrell. Low-power wide area network (lpwan) overview. https://tools.ietf.org/html/rfc8376, 2019.

[UCM14] UCM. Biblioteca complutense. https://biblioteca.ucm.es/google8, 2014.

Apéndice A

Apendice A

A.1. Instalación y configuración del gateway

Utilizando los repositorios de distribuciones oficiales de SO de Raspberry Pi, se descarga e instala la versión `lite` de `Raspbian` en la tarjeta SD que se inserte en la `Raspberry Pi`. Para establecer una conexión SSH por terminal es necesario crear un fichero con nombre `ssh` en la raíz de la unidad de almacenamiento previamente al primer arranque del SO. La distribución originalmente estaba configurada por defecto con la conexión de SSH abierta en el puerto 22, pudiendo acceder con el usuario `pi` y la contraseña `raspberry`. Este dato era ignorado por los usuarios menos experimentados y esto supuso una brecha de seguridad en todos las distribuciones que no fueron configuradas por los usuarios según las indicaciones de la propia documentación de Raspberry [Fun18]. En el primer arranque del SO de la Raspberry se establecerán las configuraciones básicas para las sucesivas conexiones SSH basadas en autenticación con clave privada.

De las estrategias disponibles para esta configuración, se crearán las claves en el equipo remoto que se conectará al gateway, entregando mediante la primera conexión SSH con terminal la clave pública y almacenando la clave privada en el equipo remoto, reduciendo así el riesgo de ser expuesta fuera del dominio local del equipo. Para disponer de flexibilidad de conexión independientemente del SO del equipo remoto, la clave privada tendrá un formato `OpenSSH`, fácil de incluir en `Windows` o `Linux` ya sea mediante conversión de la clave a formato

PPK o como fichero accesible para aplicaciones de desarrollo, transferencias de ficheros, y/o control de versiones que integran conexiones SSH configurables (GitHub, Filezilla, Eclipse, etc). Los pasos necesarios para establecer conexiones cifradas robustas pueden encontrarse en el Anexo A.2.

Se establece la capacidad del gateway para su adaptador de red WI-FI de 2.4ghz, para actuar como punto de acceso [Fouc] mediante un acceso vía usuario y contraseña, con WPA2 y gestión de claves WPA-PSK. De esta forma, se despliega un punto de acceso de red inalámbrica que permitirá a otros dispositivos incorporarse a la suite domótica. En la figura A.1 se observa una representación de despliegue de dispositivos donde los nodos, en diferentes plataformas de hardware y con distintos roles de actuadores o sensores, conectándose a la misma red inalámbrica generada por el adaptador de red WI-FI integrado en el hardware del gateway.

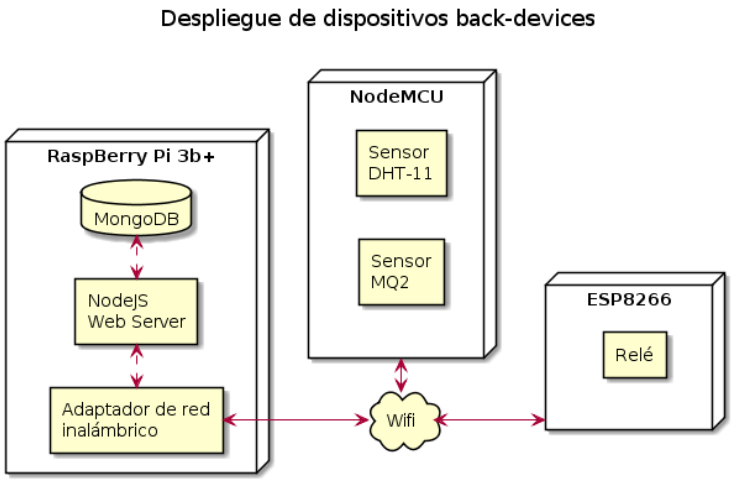

Figura A.1: *Diagrama de despliegue de dispositivos y gateway*[1]

Las placas seleccionadas para actuar como nodos del sistema están basadas en el SoC WI-FI ESP8266 y su firmware está escrito en lenguaje LUA, sin embargo, su código es habitualmente escrito en C con un alto nivel mediante herramientas de desarrollo como Arduino IDE. El proceso de subida de código en un lenguaje especifico que es transformado a otro lenguaje

y subido a una placa microcontrontroladora cuyo lenguaje es distinto para sobreescribir el firmware se conoce como `compilación cruzada` y permite una manera fácil de programar en alto nivel el código que hace funcionar una placa con SoC como las placas nodeMCU o los chips ESP8266. El código programado por el desarrollador se conoce en Arduino como `sketch`. El proceso de subida de un sketch a una placa mediante el entorno de desarrollo de Arduino se realiza mediante USB. El proceso de instalación y configuración del entorno de desarrollo de Arduino por linea de comandos, así como una primera prueba de funcionamiento del procedimiento está detallado en el Apéndice A.2. Es importante haber verificado el correcto funcionamiento de estas instrucciones antes de proseguir con el desarrollo.

Habiendo verificado el proceso de subida es necesario probar que la conexión inalámbrica y la comunicación mediante el protocolo MQTT es correcta, con un código de prueba. El proceso es el mismo que en la etapa de pruebas: Instanciar un nuevo proyecto, incluir el código, subir el sketch en la placa y probar la comunicación de la placa con el servidor. Es necesario además agregar las librerías necesarias, en el caso que ocupa este proyecto se utiliza la librería `PubSubClient de Nick O'Leary`, un cliente ligero de MQTT que posee buena estabilidad y fácil integración en desarrollo.

```
cd ~/Arduino
arduino-cli lib search PubSubClient
arduino-cli lib install "PubSubClient"
arduino-cli sketch new testmqtt
nano test/testmqtt.ino
```

El código de prueba de MQTT incluye la librería de ESP8266WiFi.h que no es necesario añadir mediante el instalador de librerías, ya que este viene incluido en el core de tarjetas de terceros que se incluyó anteriormente para el chip ESP8266. De esta forma, el código incluye los parámetros de conexión WI-FI para el adaptador de red inalámbrico del nodo principal y la conexión al servicio de MQTT así como la publicación de topics. El siguiente,

es un ejemplo básico para hacer pruebas es una ligera modificación del ejemplo contenido en la ruta `~/Arduino/libraries/PubSubClient/examples/mqtt_basic/mqtt_basic.ino`.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "edomus";
const char* password = "sistemaguardian1970.";
const char* mqtt_server = "192.168.4.1";

WiFiClient espClient;
PubSubClient client(espClient);
char msg[50];
long lastMsg = 0;
int value = 0;

void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  randomSeed(micros());
}

void callback(char* topic, byte* payload, unsigned int length) {
  // Switch on the LED if an 1 was received as first character
```

```
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);
  } else {
    digitalWrite(BUILTIN_LED, HIGH);
  }
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    // Create a random client ID
    String clientId = "nodemcuClient-test";
    clientId += String(random(0xffff), HEX);

    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      client.publish("outTopic", "hello world");
      client.subscribe("test");
    } else {
      delay(5000);
    }
  }
}

void setup() {
  pinMode(BUILTIN_LED, OUTPUT);
  setup_wifi();
```

```
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  long now = millis();

  if (now - lastMsg > 2500) {
    lastMsg = now;
    ++value;
    snprintf (msg, 50, "hello world #%ld", value);
    client.publish("test/msg", msg);
  }
}
```

Una vez subido el sketch a la placa según los pasos anteriormente indicados, podemos realizar una llamada a la placa mediante el cliente de MQTT instalado en el nodo principal. El cual generará una salida de texto sin fin. También se puede operar el led embebido en la placa con el argumento 0 o 1. Con esto, queda probado que el proceso de subida y la placa están en condiciones óptimas de funcionamiento.

```
    mosquitto_sub -h localhost -t test/msg
    mosquitto_pub -h localhost  -t test -m "1"
    mosquitto_pub -h localhost  -t test -m "0"
```

A.2. Proceso de configuración de conexion SSH para el Gateway

En Windows puede utilizarse aplicaciones de gestión de claves como 'puttygen'. En la sección de parámetros de generación de las claves se define SSH-2 RSA de 2048bits y en la sección de acciones pulsamos en 'genérate'. Tras unos movimientos aleatorios de ratón se generará la clave pública en el área de texto. Se deben guardar ambas claves mediante los botones 'save public key' y 'save private key'. Esta última será guardada con una contraseña definida en los inputs de la aplicación para tal fin. La clave privada será almacenada en formato PPK para ser rápidamente usada por aplicaciones de conexión por terminal remota como 'PUTTY'. Es recomendable exportar dicho fichero a formato OpenSSH mediante la misma aplicación de generación de claves, en la sección 'Conversions' del menú desplegable y seleccionando la opción 'Export OpenSSH key', definimos un nombre para el fichero de salida y pulsamos 'save'. Esta misma operación puede realizarse desde una terminal de un SO Linux mediante el comando `ssh-keygen -t rsa` que generará por defecto la claves en el directorio `/home/username/.ssh/` bajo el nombre `id_rsa.pub` e `id_rsa` para las claves pública y privada respectivamente.

Al no disponer de interfaz mediante dispositivos I/O para una acceso local con la Raspberry, es necesario establecer una primera conexión de terminal remoto mediante SSH con usuario y contraseña. Este primer acceso nos permite establecer las reglas de conexión que se usarán en adelante en el fichero de configuración en la ruta `/etc/ssh/sshd_config` asi como la configuracion de cuentas de usuarios.

Las distribuciones de Raspbian disponen del usuario por defecto `pi`. Esta cuenta de usuario esta incluido dentro del grupo de usuarios `sudo`. En adelante se operará con una cuenta distinta que ha de generarse manualmente y adicionalmente eliminar la cuenta

del usuario `pi` para limitar brechas de seguridad. Como primer paso, crear el usuario `sudo adduser edomus` e incluir al usuario en el grupo de usuarios `sudo`. El fichero por defecto creado durante la instalación de la distribución situado en `/etc/sudoers` dispone de la directiva `includedir /etc/sudoers.d` que debe ser descomentada en el fichero de configuración de `sudo sudoers`, mediante el comando `visudo`. Es necesario crear un fichero en la ruta `/etc/sudoers.d` con el siguiente formato de nombre `010_edomus-nopasswd` cuyo contenido incluya la siguiente linea `edomus ALL=(ALL) NOPASSWD: ALL` una vez se haya habilitado la directiva. Tras realizar las comprobaciones de que el nuevo usuario puede operar sin problemas con la nueva configuración de permisos, se elimina el fichero de permisos existente en `/etc/sudoers` para el usuario `pi`, y su eliminación del sistema con el comando `sudo deluser -remove-home pi`.

Para realizar la comunicación remota por terminal en SSH de manera más segura y cómoda, incluiremos un fichero con el contenido de la clave pública en una ruta manualmente definida dentro del 'home' del usuario edomus.

En concreto modificaremos el puerto de entrada para redirigir la conexión del puerto por defecto 22 a un valor más elevado (como por ejemplo el 45021). Esta decisión tiene como objetivo retrasar las técnicas de sondeo de puertos de un atacante hacia un servidor que admite conexiones externas. Un bot programado para encontrar servidores y marcarlos como objetivo de ataques escaneara puertos mediante evaluación de respuestas con paquetería ICMP. Igualmente un atacante puede determinar la naturaleza de los servicios ofrecidos por un servidor mediante herramientas como `Nmap`, al establecer valores elevados en los puertos, un rastreo incremental desde los valores más bajos llevará mas tiempo, permitiendo a las soluciones de seguridad (como un firewalls) del servidor detectar el ataque con margen mayor de tiempo.

En este mismo fichero establecemos unos límites concretos en los valores de tiempo de gracia `LoginGraceTime 5` de apenas 5 segundos, impedimos el acceso del usuario root desde una conexión externa `PermitRootLogin no`, limitamos el número de intentos de conexión `MaxAuthTries 3` y el número máximo de sesiones simultáneas `MaxSessions`. Para admitir las conexiones SSH mediante una autentificación con clave es necesario habilitar la autentificación de clave pública `PubkeyAuthentication yes` y definir la ruta del fichero con la clave publica almacenada localmente en el servidor `AuthorizedKeysFile .net/.aut` (véase que en este caso hemos definido una ruta manualmente indicando que la clave pública se encuentra en un fichero oculto nombrado `aut` en la ruta `/home/pi/.net`). Como refuerzo adicional configuramos el servidor para denegar todo intento de conexión mediante contraseña plana `PasswordAuthentication no`, y adicionalmente limitar el acceso sólo a las cuentas de usuarios designadas `AllowUsers edomus`. Definidos los nuevos cambios de configuración, es necesario reiniciar el servicio.

A.3. Instalación de aplicaciones y servicios

Lo primero es descargar la signing key o clave de firma utilizando el comando wget. Añadimos la clave para a una lista para autenticar el paquete que vamos a descargar más tarde.

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
sudo apt-get update
sudo apt-get install mosquitto
```

Si durante el proceso de instalación se encontrase errores de dependencias, algo común en distribuciones Raspbian, puede revisarse el apartado del apéndice B.2 de Troubleshooting.

A.4. Proceso de instalación y configuración de Arduino en linea de comandos

Al tratarse el SO del nodo principal de un entorno de terminal Command line interface (CLI), no podrá utilizarse el Integrated Development Environment (IDE) de Arduino de escritorio, que pese a disponer de compilaciones para la mayoría de distribuciones de `GNU\Linux`, requiere de un entorno gráfico para para su ejecución. Sin embargo, en 2018 la compañía de Arduino anuncio arduino-cli, que se presenta como la alternativa CLI del IDE de Arduino. Aunque sigue siendo una herramienta de software aun en desarrollo, ya dispone de las funcionalidades necesarias para compilar y subir un sketch con librerías en una placa de terceros como las nodemcu. El proceso de instalación y configuración aplicado en este proyecto se sucede de la siguiente manera.

Se dispone de alternativas adicionales de instalación como la compilación mediante Go, opción que quedo descartada tras las complicaciones de versiones de compilador que generaban error en el proceso de instalación. Por ello, se opto por la instalación manual. Primero se ha de descargar el binario ejecutable de arduino-cli para la arquitectura correspondiente. En el modelo de Raspberry Pi 3B+ se dispone de un procesador ARMv7, es importante asegurarse de comprobar que la versión descargada de la sección de releases corresponda con el equipo, esto puede verificarse en Raspbian mediante el comando `less /proc/cpuinfo`. Durante el desarrollo de este prototipo, la versión de aruduino-cli era 0.0.100.

```
wget https://github.com/arduino/arduino-cli/releases/download/{version}/{paquete}
tar -xvzf arduino-cli_0.0.100_Linux_ARMv7.tar.g
sudo mv arduino-cli /bin/ && rm arduino-cli_0.0.100_Linux_ARMv7.tar.gz
```

De esta manera, habiendo ubicado el binario en una ruta del PATH de ejecutables disponible para el usuario, se puede ejecutar el software desde cualquier ubicación. El comando arduino-cli debe ejecutarse una primera vez para generar las carpetas necesarias para le

generación de sketchs, ubicar librerías para el código y demás ficheros que permitirán subir código a las placas, se generara una carpeta en la raiz del usuario llamada Arduino, desde la cual se gestionaran los proyectos. Por defecto, arduino-cli dispone de la información necesaria para utilizar las placas genéricas de Arduino. Dado que se utilizaran placas ensambladas por terceros con el modulo WI-FI esp8266 integrado, sera necesario importar las librerías necesarias.

```
arduino-cli
cd ~/Arduino/
nano arduino-cli.yaml
```

Dentro del fichero arduino-cli.yaml es necesario ingresar la referencias de las librerías de placas de terceros. Es necesario copiar el siguiente contenido y guardar.

```
board_manager:
    additional_urls:
        - http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

A continuación se debe refrescar la lista de placas disponibles en los registros de la aplicación y verificar la identidad de la placa. En este punto se debe conectar mediante USB una placa nodeMCU a alguno de los puertos USB disponibles de la Raspberry. Para verificar que se ha conectado correctamente, pueden consultare comandos como `dmesg tail` o la ruta /dev en la cual debe aprecer el dispositivo generalmente reconocido como `ttyUSB0`, aunque dicho valor puede variar según distribución y modelo de placa de Arduino. Para poder operar sobre el dispositivo sin la necesidad de permisos de root, facilitando asi las operaciones de arduino-cli, es recomendable habilitar permisos de lectura y escritura sobre el dispositivo conectado.

```
sudo chmod a+rw /dev/ttyUSB0
arduino-cli core update-index
```

```
arduino-cli core install esp8266:esp8266
arduino-cli core update-index
arduino-cli board listall
```

En la lista resultante debe aparecer las placas externas con su denominación FQBN, dicha denominación es el argumento que se proveerá en la subida de sketch mediante arduino-cli. A continuación debe comprobarse que puede compilarse y subir correctamente un sketch sencillo para verificar que todo el proceso de instalación y configuración ha sido correcto.

```
cd ~/Arduino
arduino-cli sketch new test
nano test/test.ino
```

Agregar el siguiente codigo en el fichero y guardar.

```
void setup() {
  pinMode(2, OUTPUT);
}

void loop() {
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```

Esto creara una carpeta de proyecto con el correspondiente fichero .ino incluyendo el código a subir en la placa, este test básico consiste en montar un led que parpadee a intervalos de 1 segundo siguiendo el siguiente montaje:

A continuación debe compilarse el proyecto y subirse a la placa, una vez terminado el proceso, el led debe parpadear según lo programado:

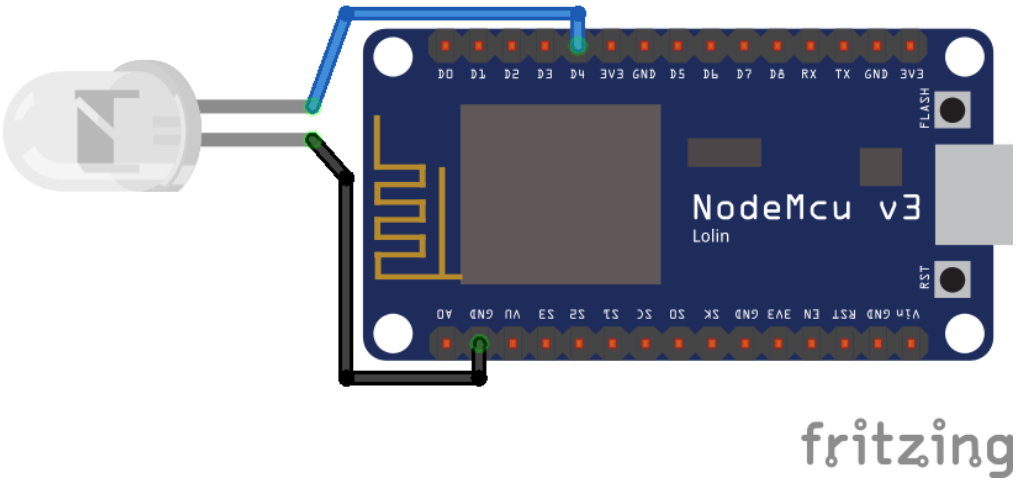

Figura A.2: *Esquema de montaje de led en una placa nodemcu*

```
arduino-cli compile --fqbn esp8266:esp8266:nodemcuv2 test/test
arduino-cli upload -p /dev/ttyUSB0 --fqbn esp8266:esp8266:nodemcuv2 test/test
```

Apéndice B

Troubleshooting

B.1. Proceso de subida de sckets pacas nodeMCU desde Raspberry Pi

Para las pruebas de compilación de arduino-cli:

sudo apt-get install git raspberrypi-kernel-headers build-essential dkms

Descargar el driver correcto: https://github.com/juliagoda/CH341SER

Seleccionar placas nano328

B.2. Error en la instalación de mosquitto

Durante el proceso de instalación de de la aplicación de `mosquitto` para realizar pruebas de comunicación del protocolo MQTT entre el gateway y los nodos, es posible que se muestre el siguiente error:

The following packages have unmet dependencies:

mosquitto : Depends: libssl1.0.0 (>= 1.0.0) but it is not installable

Depends: libwebsockets3 (>= 1.2) but it is not installable

La solución para este problema puede encontrarse en el siguiente enlace:

https://theembeddedlab.com/tutorials/install-mosquitto-on-a-raspberry-pi/

B.3. Baja calidad de la señal Wi-FI en los nodos

Debe tenerse en cuenta que las especificaciones del modulo esp8266 de wifi requiere de una alimentación de 3.3V que pueden ser suministrados por la placa microcontroladora como es el caso de las placas `nodeMCU`, sin embargo, en las placas de SoC WI-FI ESP8266 esto deja un problema de intensidad en la alimentación del modulo, ya que el pin de 3.3V disponible en la placa posees un amperaje de 50mA y se requieren de unos 200mA para garantizar una comunicación estable. Puede reforzarse la alimentación mediante el soldado de pines en la placa, pero este procedimiento debe seguirse bajo las indicaciones de un tutorial claro y que demuestre los resultados del proceso.

B.4. Margenes de error del sensor DHT11

sensor DHT version 1.4. Se debe realizar una prueba de contacto con un script que nos muestre la información por pantalla para verificar que las conexiones del sensor a la raps-berry son correctos. Este primer script es un bucle infinito de mediciones de temperatura y humedad cada poco segundos. Hay una justificación para elegir un bucle sobre una única medida del sensor y está fundamentada en el margen de error inicial de las medidas. Si bien el sensor DTH11 es una opción muy común por su bajo coste y facilidad de implementación (este sensor se caracteriza por tener la señal digital calibrada por lo que asegura una alta calidad y una fiabilidad a lo largo del tiempo, ya que contiene un microcontrolador de 8 bits integrado. Está constituido por dos sensores resistivos (NTC y humedad) - revisar esta info y contrastarla contra ésta: https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/), ademas de manejar señales digitales que no se ven afectadas por las fluctuaciones de voltaje, tiene algunas contrapartidas que deben tenerse en cuenta. Se necesita un tiempo mínimo de espera entre medidas (de al menos 1 segundo), hecho que no agrava particularmente su desempeño en entornos cerrados como una casa, ya que las variaciones de temperatura y humedad no son bruscas, aun así existen estrategias para reducir estos tiempos, por ejemplo, usar la función millis() de

Arduino, el cual nos da el tiempo en milisegundos desde que empieza a ejecutarse el código, De esta forma evitamos la pausa de los 2 segundos, pero no el tiempo que demora en hacer la lectura, que es de aproximadamente 250 milisegundos, el cual lo pueden notar si realizan el ejemplo anterior, en donde se hace parpadear el led interno de la placa (Pin 13) con pausas de 100ms (tomado de https://naylampmechatronics.com/blog/40_Tutorial-sensor-de-temperatura-y-humedad-DHT1.html). Otro problema que abordar es que las primeras lecturas tienen un margen de error de unos +-2 grados Celsius y +-5 % de humedad relativa en las primeras 4 lecturas. Esto generará un problema a la hora de tomar lecturas instantáneas si el sensor no se encuentra ya operando cuando se solicita el dato.

Acrónimos

6lowpan	IPv6 over Low power Wireless Personal Area Networks.
AEPD	Agencia española de protección de datos.
API	Application Programming Interface.
BBDD	Base de datos.
CLI	Command line interface.
DCP	Datos de carácter personal.
GPIO	General Purpose Input/Output.
HTTP	HyperText Transfer Protocol Secure.
HTTP	HyperText Transfer Protocol.
I/O	Imput/Output.
IDE	Integrated Development Environment.
IoE	Internet of Everything.
IoT	Internet de las Cosas.
ISP	Proveedor de servicios de internet.
JSON	JavaScript Object Notation.
MQTT	Message Queuing Telemetry Transport.
PaaS	Plataforma como servicio.
QOS	Quality of Service.
RGPD	Reglamento general de protección de datos.
SO	Sistema Operativo.
SoC	IPv6 over Low power Wireless Personal Area Networks.
SSH	Secure SHell.

TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
UDP	User Datagram Protocol.
USB	Universal Serial Bus.
WI-FI	Wireless Fidelity.
WLAN	wireless local area network.